Internet Engineering Task Force (IETF)
Request for Comments: 9053
Obsoletes: 8152
Category: Informational
ISSN: 2070-1721

J. Schaad
August Cellars
August 2022

CBOR Object Signing and Encryption (COSE): Initial Algorithms

Abstract

Concise Binary Object Representation (CBOR) is a data format designed for small code size and small message size. There is a need to be able to define basic security services for this data format. This document defines a set of algorithms that can be used with the CBOR Object Signing and Encryption (COSE) protocol (RFC 9052).

This document, along with RFC 9052, obsoletes RFC 8152.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There has been an increased focus on small, constrained devices that make up the Internet of Things (IoT). One of the standards that has come out of this process is "Concise Binary Object Representation (CBOR)" [STD94]. CBOR extended the data model of JavaScript Object Notation (JSON) [STD90] by allowing for binary data, among other changes. CBOR has been adopted by several of the IETF working groups dealing with the IoT world as their method of encoding data structures. CBOR was designed specifically to be small in terms of both messages transported and implementation size and to have a schema-free decoder. A need exists to provide message security services for IoT, and using CBOR as the message-encoding format makes sense.

The core COSE specification consists of two documents. [RFC9052] contains the serialization structures and the procedures for using

the different cryptographic algorithms. This document provides an initial set of algorithms for use with those structures.

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. Changes from RFC 8152

* Extracted the sections dealing with specific algorithms and placed them into this document. The sections dealing with structure and general processing rules are placed in [RFC9052].

* Made text clarifications and changes in terminology.

* Removed all of the details relating to countersignatures and placed them in [COUNTERSIGN].

1.3. Document Terminology

In this document, we use the following terminology:

Byte: A synonym for octet.

Constrained Application Protocol (CoAP): A specialized web transfer protocol for use in constrained systems. It is defined in [RFC7252].

Authenticated Encryption (AE) algorithms [RFC5116]: Encryption algorithms that provide an authentication check of the contents along with the encryption service. An example of an AE algorithm used in COSE is AES Key Wrap [RFC3394]. These algorithms are used for key encryption, but Authenticated Encryption with Associated Data (AEAD) algorithms would be preferred.

AEAD algorithms [RFC5116]: Encryption algorithms that provide the same authentication service of the content as AE algorithms do, and also allow associated data that is not part of the encrypted body to be included in the authentication service. An example of an AEAD algorithm used in COSE is AES-GCM [RFC5116]. These algorithms are used for content encryption and can be used for key encryption as well.

The term "byte string" is used for sequences of bytes, while the term "text string" is used for sequences of characters.

The tables for algorithms contain the following columns:

* A name for the algorithm for use in documents.

* The value used on the wire for the algorithm. One place this is used is the algorithm header parameter of a message.

* A short description so that the algorithm can be easily identified when scanning the IANA registry.

Additional columns may be present in a table depending on the algorithms.

1.4. CDDL Grammar for CBOR Data Structures

When COSE was originally written, the Concise Data Definition

Language (CDDL) [RFC8610] had not yet been published in an RFC, so it could not be used as the data description language to normatively describe the CBOR data structures employed by COSE. For that reason, the CBOR data objects defined here are described in prose. Additional (non-normative) descriptions of the COSE data objects are provided in a subset of CDDL, described in [RFC9052].

1.5. Examples

A GitHub project has been created at [GitHub-Examples] that contains a set of testing examples. Each example is found in a JSON file that contains the inputs used to create the example, some of the intermediate values that can be used for debugging, and the output of the example. The results are encoded using both hexadecimal and CBOR diagnostic notation format.

Some of the examples are designed to be failure-testing cases; these are clearly marked as such in the JSON file.

2. Signature Algorithms

Section 8.1 of [RFC9052] contains a generic description of signature algorithms. This document defines signature algorithm identifiers for two signature algorithms.

2.1. ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) [DSS] defines a signature algorithm using Elliptic Curve Cryptography (ECC). Implementations SHOULD use a deterministic version of ECDSA such as the one defined in [RFC6979]. The use of a deterministic signature algorithm allows systems to avoid relying on random number generators in order to avoid generating the same value of "k" (the per-message random value). Biased generation of the value "k" can be attacked, and collisions of this value lead to leaked keys. It additionally allows performing deterministic tests for the signature algorithm. The use of deterministic ECDSA does not lessen the need to have good random number generation when creating the private key.

The ECDSA signature algorithm is parameterized with a hash function (h). In the event that the length of the hash function output is greater than the group of the key, the leftmost bytes of the hash output are used.

The algorithms defined in this document can be found in Table 1.

Name	Value	Hash	Description
ES256	-7	SHA-256	ECDSA w/ SHA-256
ES384	-35	SHA-384	ECDSA w/ SHA-384
ES512	-36	SHA-512	ECDSA w/ SHA-512

Table 1: ECDSA Algorithm Values

This document defines ECDSA as working only with the curves P-256, P-384, and P-521. This document requires that the curves be encoded using the "EC2" (two coordinate elliptic curve) key type. Implementations need to check that the key type and curve are correct when creating and verifying a signature. Future documents may define it to work with other curves and key types in the future.

In order to promote interoperability, it is suggested that SHA-256 be

used only with curve P-256, SHA-384 be used only with curve P-384, and SHA-512 be used only with curve P-521. This is aligned with the recommendation in Section 4 of [RFC5480].

The signature algorithm results in a pair of integers (R, S). These integers will be the same length as the length of the key used for the signature process. The signature is encoded by converting the integers into byte strings of the same length as the key size. The length is rounded up to the nearest byte and is left padded with zero bits to get to the correct length. The two integers are then concatenated together to form a byte string that is the resulting signature.

Using the function defined in [RFC8017], the signature is:

Signature = I2OSP(R, n) | I2OSP(S, n)

where n = ceiling(key_length / 8)

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "EC2".

* If the "alg" field is present, it MUST match the ECDSA signature algorithm being used.

* If the "key_ops" field is present, it MUST include "sign" when creating an ECDSA signature.

* If the "key_ops" field is present, it MUST include "verify" when verifying an ECDSA signature.

2.1.1. Security Considerations for ECDSA

The security strength of the signature is no greater than the minimum of the security strength associated with the bit length of the key and the security strength of the hash function.

Note: Use of a deterministic signature technique is a good idea even when good random number generation exists. Doing so both reduces the possibility of having the same value of "k" in two signature operations and allows for reproducible signature values, which helps testing. There have been recent attacks involving faulting the device in order to extract the key. This can be addressed by combining both randomness and determinism [CFRG-DET-SIGS].

There are two substitution attacks that can theoretically be mounted against the ECDSA signature algorithm.

* Changing the curve used to validate the signature: If one changes the curve used to validate the signature, then potentially one could have two messages with the same signature, each computed under a different curve. The only requirements on the new curve are that its order be the same as the old one and that it be acceptable to the client. An example would be to change from using the curve secp256r1 (aka P-256) to using secp256k1. (Both are 256-bit curves.) We currently do not have any way to deal with this version of the attack except to restrict the overall set of curves that can be used.

* Changing the hash function used to validate the signature: If one either has two different hash functions of the same length or can truncate a hash function, then one could potentially find collisions between the hash functions rather than within a single hash function. For example, truncating SHA-512 to 256 bits might

collide with a SHA-256 bit hash value. As the hash algorithm is part of the signature algorithm identifier, this attack is mitigated by including a signature algorithm identifier in the protected-header bucket.

2.2. Edwards-Curve Digital Signature Algorithm (EdDSA)

[RFC8032] describes the elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (EdDSA). In that document, the signature algorithm is instantiated using parameters for the edwards25519 and edwards448 curves. The document additionally describes two variants of the EdDSA algorithm: Pure EdDSA, where no hash function is applied to the content before signing, and HashEdDSA, where a hash function is applied to the content before signing and the result of that hash function is signed. For EdDSA, the content to be signed (either the message or the prehash value) is processed twice inside of the signature algorithm. For use with COSE, only the pure EdDSA version is used. This is because it is not expected that extremely large contents are going to be needed and, based on the arrangement of the message structure, the entire message is going to need to be held in memory in order to create or verify a signature. Therefore, there does not appear to be a need to be able to do block updates of the hash, followed by eliminating the message from memory. Applications can provide the same features by defining the content of the message as a hash value and transporting the COSE object (with the hash value) and the content as separate items.

The algorithm defined in this document can be found in Table 2. A single signature algorithm is defined, which can be used for multiple curves.

Name	Value	Description
EdDSA	-8	EdDSA

Table 2: EdDSA Algorithm Value

[RFC8032] describes the method of encoding the signature value.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "OKP" (Octet Key Pair).

* The "crv" field MUST be present, and it MUST be a curve defined for this signature algorithm.

* If the "alg" field is present, it MUST match "EdDSA".

* If the "key_ops" field is present, it MUST include "sign" when creating an EdDSA signature.

* If the "key_ops" field is present, it MUST include "verify" when verifying an EdDSA signature.

2.2.1. Security Considerations for EdDSA

Public values are computed differently in EdDSA and Elliptic Curve Diffie-Hellman (ECDH); for this reason, the public key from one should not be used with the other algorithm.

If batch signature verification is performed, a well-seeded cryptographic random number generator is REQUIRED (Section 8.2 of

[RFC8032]). Signing and nonbatch signature verification are deterministic operations and do not need random numbers of any kind.

3. Message Authentication Code (MAC) Algorithms

Section 8.2 of [RFC9052] contains a generic description of MAC algorithms. This section defines the conventions for two MAC algorithms.

3.1. Hash-Based Message Authentication Codes (HMACs)

HMAC [RFC2104] [RFC4231] was designed to deal with length extension attacks. The HMAC algorithm was also designed to allow new hash functions to be directly plugged in without changes to the hash function. The HMAC design process has been shown to be solid; although the security of hash functions such as MD5 has decreased over time, the security of HMAC combined with MD5 has not yet been shown to be compromised [RFC6151].

The HMAC algorithm is parameterized by an inner and outer padding, a hash function (h), and an authentication tag value length. For this specification, the inner and outer padding are fixed to the values set in [RFC2104]. The length of the authentication tag corresponds to the difficulty of producing a forgery. For use in constrained environments, we define one HMAC algorithm that is truncated. There are currently no known issues with truncation; however, the security strength of the message tag is correspondingly reduced in strength. When truncating, the leftmost tag-length bits are kept and transmitted.

The algorithms defined in this document can be found in Table 3.

Name	Value	Hash	Tag Length	Description
HMAC 256/64	4	SHA-256	64	HMAC w/ SHA-256 truncated to 64 bits
HMAC 256/256	5	SHA-256	256	HMAC w/ SHA-256
HMAC 384/384	6	SHA-384	384	HMAC w/ SHA-384
HMAC 512/512	7	SHA-512	512	HMAC w/ SHA-512

Table 3: HMAC Algorithm Values

Some recipient algorithms transport the key, while others derive a key from secret data. For those algorithms that transport the key (such as AES Key Wrap), the size of the HMAC key SHOULD be the same size as the output of the underlying hash function. For those algorithms that derive the key (such as ECDH), the derived key MUST be the same size as the output of the underlying hash function.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the HMAC algorithm being used.

* If the "key_ops" field is present, it MUST include "MAC create"

when creating an HMAC authentication tag.

* If the "key_ops" field is present, it MUST include "MAC verify" when verifying an HMAC authentication tag.

Implementations creating and validating MAC values MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

3.1.1. Security Considerations for HMAC

HMAC has proved to be resistant to attack even when used with weakened hash algorithms. The current best known attack is to brute force the key. This means that key size is going to be directly related to the security of an HMAC operation.

3.2. AES Message Authentication Code (AES-CBC-MAC)

AES-CBC-MAC is the instantiation of the CBC-MAC construction (defined in [MAC]) using AES as the block cipher. For brevity, we also use "AES-MAC" to refer to AES-CBC-MAC. (Note that this is not the same algorithm as AES Cipher-Based Message Authentication Code (AES-CMAC) [RFC4493].)

AES-CBC-MAC is parameterized by the key length, the authentication tag length, and the Initialization Vector (IV) used. For all of these algorithms, the IV is fixed to all zeros. We provide an array of algorithms for various key and tag lengths. The algorithms defined in this document are found in Table 4.

Name	Value	Key Length	Tag Length	Description
AES-MAC 128/64	14	128	64	AES-MAC 128-bit key, 64-bit tag
AES-MAC 256/64	15	256	64	AES-MAC 256-bit key, 64-bit tag
AES-MAC 128/128	25	128	128	AES-MAC 128-bit key, 128-bit tag
AES-MAC 256/128	26	256	128	AES-MAC 256-bit key, 128-bit tag

Table 4: AES-MAC Algorithm Values

Keys may be obtained from either a key structure or a recipient structure. Implementations creating and validating MAC values MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the AES-MAC algorithm being used.

* If the "key_ops" field is present, it MUST include "MAC create" when creating an AES-MAC authentication tag.

* If the "key_ops" field is present, it MUST include "MAC verify" when verifying an AES-MAC authentication tag.

3.2.1. Security Considerations for AES-CBC-MAC

A number of attacks exist against Cipher Block Chaining Message Authentication Code (CBC-MAC) that need to be considered.

* A single key must only be used for messages of a fixed or known length. If this is not the case, an attacker will be able to generate a message with a valid tag given two message and tag pairs. This can be addressed by using different keys for messages of different lengths. The current structure mitigates this problem, as a specific encoding structure that includes lengths is built and signed. (CMAC also addresses this issue.)

* In Cipher Block Chaining (CBC) mode, if the same key is used for both encryption and authentication operations, an attacker can produce messages with a valid authentication code.

* If the IV can be modified, then messages can be forged. This is addressed by fixing the IV to all zeros.

4. Content Encryption Algorithms

Section 8.3 of [RFC9052] contains a generic description of content encryption algorithms. This document defines the identifier and usages for three content encryption algorithms.

4.1. AES-GCM

The Galois/Counter Mode (GCM) mode is a generic AEAD block cipher mode defined in [AES-GCM]. The GCM mode is combined with the AES block encryption algorithm to define an AEAD cipher.

The GCM mode is parameterized by the size of the authentication tag and the size of the nonce. This document fixes the size of the nonce at 96 bits. The size of the authentication tag is limited to a small set of values. For this document, however, the size of the authentication tag is fixed at 128 bits.

The set of algorithms defined in this document is in Table 5.

Name	Value	Description
A128GCM	1	AES-GCM mode w/ 128-bit key, 128-bit tag
A192GCM	2	AES-GCM mode w/ 192-bit key, 128-bit tag
A256GCM	3	AES-GCM mode w/ 256-bit key, 128-bit tag

Table 5: Algorithm Values for AES-GCM

Keys may be obtained from either a key structure or a recipient structure. Implementations that are encrypting or decrypting MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the AES-GCM algorithm being used.

* If the "key_ops" field is present, it MUST include "encrypt" or "wrap key" when encrypting.

* If the "key_ops" field is present, it MUST include "decrypt" or "unwrap key" when decrypting.

4.1.1. Security Considerations for AES-GCM

When using AES-GCM, the following restrictions MUST be enforced:

* The key and nonce pair MUST be unique for every message encrypted.

* The total number of messages encrypted for a single key MUST NOT exceed 2^32 [SP800-38D]. An explicit check is required only in environments where it is expected that this limit might be exceeded.

* [RFC8446] contains an analysis on the use of AES-CGM for its environment. Based on that recommendation, one should restrict the number of messages encrypted to 2^24.5.

* A more recent analysis in [ROBUST] indicates that the number of failed decryptions needs to be taken into account as part of determining when a key rollover is to be done. Following the recommendation in DTLS (Section 4.5.3 of [RFC9147]), the number of failed message decryptions should be limited to 2^36.

Consideration was given to supporting smaller tag values; the constrained community would desire tag sizes in the 64-bit range. Such use drastically changes both the maximum message size (generally not an issue) and the number of times that a key can be used. Given that Counter with CBC-MAC (CCM) is the usual mode for constrained environments, restricted modes are not supported.

4.2. AES-CCM

CCM is a generic authentication encryption block cipher mode defined in [RFC3610]. The CCM mode is combined with the AES block encryption algorithm to define an AEAD cipher that is commonly used in constrained devices.

The CCM mode has two parameter choices. The first choice is M, the size of the authentication field. The choice of the value for M involves a trade-off between message growth (from the tag) and the probability that an attacker can undetectably modify a message. The second choice is L, the size of the length field. This value requires a trade-off between the maximum message size and the size of the nonce.

It is unfortunate that the specification for CCM specified L and M as a count of bytes rather than a count of bits. This leads to possible misunderstandings where AES-CCM-8 is frequently used to refer to a version of CCM mode where the size of the authentication is 64 bits and not 8 bits. In most cryptographic algorithm specifications, these values have traditionally been specified as bit counts rather than byte counts. This document will follow the convention of using bit counts so that it is easier to compare the different algorithms presented in this document.

We define a matrix of algorithms in this document over the values of L and M. Constrained devices are usually operating in situations where they use short messages and want to avoid doing recipient-specific cryptographic operations. This favors smaller values of both L and M. Less-constrained devices will want to be able to use larger messages and are more willing to generate new keys for every operation. This favors larger values of L and M.

The following values are used for L:

16 bits (2): This limits messages to 2^16 bytes (64 KiB) in length. This is sufficiently long for messages in the constrained world. The nonce length is 13 bytes allowing for 2^104 possible values of the nonce without repeating.

64 bits (8): This limits messages to 2^64 bytes in length. The nonce length is 7 bytes, allowing for 2^56 possible values of the nonce without repeating.

The following values are used for M:

64 bits (8): This produces a 64-bit authentication tag. This implies that there is a 1 in 2^64 chance that a modified message will authenticate.

128 bits (16): This produces a 128-bit authentication tag. This implies that there is a 1 in 2^128 chance that a modified message will authenticate.

Name	Value	L	M	Key Length	Description
AES-CCM-16-64-128	10	16	64	128	AES-CCM mode 128-bit key, 64-bit tag, 13-byte nonce
AES-CCM-16-64-256	11	16	64	256	AES-CCM mode 256-bit key, 64-bit tag, 13-byte nonce
AES-CCM-64-64-128	12	64	64	128	AES-CCM mode 128-bit key, 64-bit tag, 7-byte nonce
AES-CCM-64-64-256	13	64	64	256	AES-CCM mode 256-bit key, 64-bit tag, 7-byte nonce
AES-CCM-16-128-128	30	16	128	128	AES-CCM mode 128-bit key, 128-bit tag, 13-byte nonce
AES-CCM-16-128-256	31	16	128	256	AES-CCM mode 256-bit key, 128-bit tag, 13-byte nonce
AES-CCM-64-128-128	32	64	128	128	AES-CCM mode 128-bit key, 128-bit tag, 7-byte nonce
AES-CCM-64-128-256	33	64	128	256	AES-CCM mode 256-bit key, 128-bit tag, 7-byte nonce

Table 6: Algorithm Values for AES-CCM

Keys may be obtained from either a key structure or a recipient structure. Implementations that are encrypting or decrypting MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the AES-CCM algorithm being used.

* If the "key_ops" field is present, it MUST include "encrypt" or "wrap key" when encrypting.

* If the "key_ops" field is present, it MUST include "decrypt" or "unwrap key" when decrypting.

4.2.1. Security Considerations for AES-CCM

When using AES-CCM, the following restrictions MUST be enforced:

* The key and nonce pair MUST be unique for every message encrypted. Note that the value of L influences the number of unique nonces.

* The total number of times the AES block cipher is used MUST NOT exceed 2^61 operations. This limit is the sum of times the block cipher is used in computing the MAC value and performing stream encryption operations. An explicit check is required only in environments where it is expected that this limit might be exceeded.

* [RFC9147] contains an analysis on the use of AES-CCM for its environment. Based on that recommendation, one should restrict the number of messages encrypted to 2^23.

* In addition to the number of messages successfully decrypted, the number of failed decryptions needs to be tracked as well. Following the recommendation in DTLS (Section 4.5.3 of [RFC9147]), the number of failed message decryptions should be limited to 2^23.5. If one is using the 64-bit tag, then the limits are significantly smaller if one wants to keep the same integrity limits. A protocol recommending this needs to analyze what level of integrity is acceptable for the smaller tag size. It may be that, to keep the desired level of integrity, one needs to rekey as often as every 2^7 messages.

[RFC3610] additionally calls out one other consideration of note. It is possible to do a precomputation attack against the algorithm in cases where portions of the plaintext are highly predictable. This reduces the security of the key size by half. Ways to deal with this attack include adding a random portion to the nonce value and/or increasing the key size used. Using a portion of the nonce for a random value will decrease the number of messages that a single key can be used for. Increasing the key size may require more resources in the constrained device. See Sections 5 and 10 of [RFC3610] for more information.

4.3. ChaCha20 and Poly1305

ChaCha20 and Poly1305 combined together is an AEAD mode that is defined in [RFC8439]. This is an algorithm defined using a cipher

that is not AES and thus would not suffer from any future weaknesses found in AES. These cryptographic functions are designed to be fast in software-only implementations.

The ChaCha20/Poly1305 AEAD construction defined in [RFC8439] has no parameterization. It takes as inputs a 256-bit key and a 96-bit nonce, as well as the plaintext and additional data, and produces the ciphertext as an output. We define one algorithm identifier for this algorithm in Table 7.

Name	Value	Description
ChaCha20/Poly1305	24	ChaCha20/Poly1305 w/ 256-bit key, 128-bit tag

Table 7: Algorithm Value for ChaCha20/Poly1305

Keys may be obtained from either a key structure or a recipient structure. Implementations that are encrypting or decrypting MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the ChaCha20/Poly1305 algorithm being used.

* If the "key_ops" field is present, it MUST include "encrypt" or "wrap key" when encrypting.

* If the "key_ops" field is present, it MUST include "decrypt" or "unwrap key" when decrypting.

4.3.1. Security Considerations for ChaCha20/Poly1305

The key and nonce values MUST be a unique pair for every invocation of the algorithm. Nonce counters are considered to be an acceptable way of ensuring that they are unique.

A more recent analysis in [ROBUST] indicates that the number of failed decryptions needs to be taken into account as part of determining when a key rollover is to be done. Following the recommendation in DTLS (Section 4.5.3 of [RFC9147]), the number of failed message decryptions should be limited to 2^36.

[RFC8446] notes that the (64-bit) record sequence number would wrap before the safety limit is reached for ChaCha20/Poly1305. COSE implementations should not send more than 2^64 messages encrypted using a single ChaCha20/Poly1305 key.

5. Key Derivation Functions (KDFs)

Section 8.4 of [RFC9052] contains a generic description of key derivation functions. This document defines a single context structure and a single KDF. These elements are used for all of the recipient algorithms defined in this document that require a KDF process. These algorithms are defined in Sections 6.1.2, 6.3.1, and 6.4.1.

5.1. HMAC-Based Extract-and-Expand Key Derivation Function (HKDF)

The HKDF key derivation algorithm is defined in [RFC5869] and [HKDF].

The HKDF algorithm takes these inputs:

secret: A shared value that is secret. Secrets may be either previously shared or derived from operations like a Diffie-Hellman (DH) key agreement.

salt: An optional value that is used to change the generation process. The salt value can be either public or private. If the salt is public and carried in the message, then the "salt" algorithm header parameter defined in Table 9 is used. While [RFC5869] suggests that the length of the salt be the same as the length of the underlying hash value, any positive salt length will improve the security, as different key values will be generated. This parameter is protected by being included in the key computation and does not need to be separately authenticated. The salt value does not need to be unique for every message sent.

length: The number of bytes of output that need to be generated.

context information: Information that describes the context in which the resulting value will be used. Making this information specific to the context in which the material is going to be used ensures that the resulting material will always be tied to that usage. The context structure defined in Section 5.2 is used by the KDFs in this document.

PRF: The underlying pseudorandom function to be used in the HKDF algorithm. The PRF is encoded into the HKDF algorithm selection.

HKDF is defined to use HMAC as the underlying PRF. However, it is possible to use other functions in the same construct to provide a different KDF that is more appropriate in the constrained world. Specifically, one can use AES-CBC-MAC as the PRF for the expand step, but not for the extract step. When using a good random shared secret of the correct length, the extract step can be skipped. For the AES algorithm versions, the extract step is always skipped.

The extract step cannot be skipped if the secret is not uniformly random -- for example, if it is the result of an ECDH key agreement step. This implies that the AES HKDF version cannot be used with ECDH. If the extract step is skipped, the "salt" value is not used as part of the HKDF functionality.

The algorithms defined in this document are found in Table 8.

Name	PRF	Description
HKDF SHA-256	HMAC with SHA-256	HKDF using HMAC SHA-256 as the PRF
HKDF SHA-512	HMAC with SHA-512	HKDF using HMAC SHA-512 as the PRF
HKDF AES-MAC-128	AES-CBC-MAC-128	HKDF using AES-MAC as the PRF w/ 128-bit key
HKDF AES-MAC-256	AES-CBC-MAC-256	HKDF using AES-MAC as the PRF w/ 256-bit key

Table 8: HKDF Algorithms

+======+=======+======+==========+=========+

Name	Label	Type	Algorithm	Description
salt	-20	bstr	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	Random salt

Table 9: HKDF Algorithm Parameters

5.2. Context Information Structure

The context information structure is used to ensure that the derived keying material is "bound" to the context of the transaction. The context information structure used here is based on that defined in [SP800-56A]. By using CBOR for the encoding of the context information structure, we automatically get the same type and length separation of fields that is obtained by the use of ASN.1. This means that there is no need to encode the lengths for the base elements, as it is done by the encoding used in JSON Object Signing and Encryption (JOSE) (Section 4.6.2 of [RFC7518]).

The context information structure refers to PartyU and PartyV as the two parties that are doing the key derivation. Unless the application protocol defines differently, we assign PartyU to the entity that is creating the message and PartyV to the entity that is receiving the message. By defining this association, different keys will be derived for each direction, as the context information is different in each direction.

The context structure is built from information that is known to both entities. This information can be obtained from a variety of sources:

* Fields can be defined by the application. This is commonly used to assign fixed names to parties, but it can be used for other items such as nonces.

* Fields can be defined by usage of the output. Examples of this are the algorithm and key size that are being generated.

* Fields can be defined by parameters from the message. We define a set of header parameters in Table 10 that can be used to carry the values associated with the context structure. Examples of this are identities and nonce values. These header parameters are designed to be placed in the unprotected bucket of the recipient structure; they do not need to be in the protected bucket, since they are already included in the cryptographic computation by virtue of being included in the context structure.

Name	Label	Type	Algorithm	Description
PartyU identity	-21	bstr	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512,	PartyU identity information

			ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	
PartyU nonce	-22	bstr / int	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	PartyU provided nonce
PartyU other	-23	bstr	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	PartyU other provided information
PartyV identity	-24	bstr	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	PartyV identity information
PartyV nonce	-25	bstr / int	direct+HKDF-SHA-256, direct+HKDF-SHA-512, direct+HKDF-AES-128, direct+HKDF-AES-256, ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW, ECDH-SS+A128KW,	PartyV provided nonce

			ECDH-SS+A192KW,	
			ECDH-SS+A256KW	
PartyV other	-26	bstr	direct+HKDF-SHA-256,	PartyV other provided information
			direct+HKDF-SHA-512,	
			direct+HKDF-AES-128,	
			direct+HKDF-AES-256,	
			ECDH-ES+HKDF-256,	
			ECDH-ES+HKDF-512,	
			ECDH-SS+HKDF-256,	
			ECDH-SS+HKDF-512,	
			ECDH-ES+A128KW,	
			ECDH-ES+A192KW,	
			ECDH-ES+A256KW,	
			ECDH-SS+A128KW,	
			ECDH-SS+A192KW,	
			ECDH-SS+A256KW	

Table 10: Context Algorithm Parameters

We define a CBOR object to hold the context information. This object is referred to as COSE_KDF_Context. The object is based on a CBOR array type. The fields in the array are:

AlgorithmID: This field indicates the algorithm for which the key material will be used. This normally is either a key wrap algorithm identifier or a content encryption algorithm identifier. The values are from the "COSE Algorithms" registry. This field is required to be present. The field exists in the context information so that a different key is generated for each algorithm even if all of the other context information is the same. In practice, this means if algorithm A is broken and thus finding the key is relatively easy, the key derived for algorithm B will not be the same as the key derived for algorithm A.

PartyUInfo: This field holds information about PartyU. The PartyUInfo is encoded as a CBOR array. The elements of PartyUInfo are encoded in the order presented below. The elements of the PartyUInfo array are:

identity: This contains the identity information for PartyU. The identities can be assigned in one of two manners. First, a protocol can assign identities based on roles. For example, the roles of "client" and "server" may be assigned to different entities in the protocol. Each entity would then use the correct label for the data it sends or receives. The second way for a protocol to assign identities is to use a name based on a naming system (i.e., DNS or X.509 names).

We define an algorithm parameter, "PartyU identity", that can be used to carry identity information in the message. However, identity information is often known as part of the protocol and can thus be inferred rather than made explicit. If identity information is carried in the message, applications SHOULD have a way of validating the supplied identity information. The identity information does not need to be specified and is set to nil in that case.

nonce: This contains a nonce value. The nonce can be either implicit from the protocol or carried as a value in the unprotected header bucket.

We define an algorithm parameter, "PartyU nonce", that can be used to carry this value in the message; however, the nonce value could be determined by the application and its value

obtained in a different manner.

This option does not need to be specified; if not needed, it is set to nil.

other: This contains other information that is defined by the protocol. This option does not need to be specified; if not needed, it is set to nil.

PartyVInfo: This field holds information about PartyV. The content of the structure is the same as for the PartyUInfo but for PartyV.

SuppPubInfo: This field contains public information that is mutually known to both parties, and is encoded as a CBOR array.

keyDataLength: This is set to the number of bits of the desired output value. This practice means if algorithm A can use two different key lengths, the key derived for the longer key size will not contain the key for the shorter key size as a prefix.

protected: This field contains the protected parameter field. If there are no elements in the "protected" field, then use a zero-length bstr.

other: This field is for free-form data defined by the application. For example, an application could define two different byte strings to be placed here to generate different keys for a data stream versus a control stream. This field is optional and will only be present if the application defines a structure for this information. Applications that define this SHOULD use CBOR to encode the data so that types and lengths are correctly included.

SuppPrivInfo: This field contains private information that is mutually known private information. An example of this information would be a pre-existing shared secret. (This could, for example, be used in combination with an ECDH key agreement to provide a secondary proof of identity.) The field is optional and will only be present if the application defines a structure for this information. Applications that define this SHOULD use CBOR to encode the data so that types and lengths are correctly included.

The following CDDL fragment corresponds to the text above.

```
PartyInfo = (
    identity : bstr / nil,
    nonce : bstr / int / nil,
    other : bstr / nil
)

COSE_KDF_Context = [
    AlgorithmID : int / tstr,
    PartyUInfo : [ PartyInfo ],
    PartyVInfo : [ PartyInfo ],
    SuppPubInfo : [
        keyDataLength : uint,
        protected : empty_or_serialized_map,
        ? other : bstr
    ],
    ? SuppPrivInfo : bstr
]
```

6. Content Key Distribution Methods

Section 8.5 of [RFC9052] contains a generic description of content

key distribution methods. This document defines the identifiers and usage for a number of content key distribution methods.

6.1. Direct Encryption

A direct encryption algorithm is defined in Section 8.5.1 of [RFC9052]. Information about how to fill in the COSE_Recipient structure is detailed there.

6.1.1. Direct Key

This recipient algorithm is the simplest; the identified key is directly used as the key for the next layer down in the message. There are no algorithm parameters defined for this algorithm. The algorithm identifier value is assigned in Table 11.

When this algorithm is used, the "protected" field MUST be zero length. The key type MUST be "Symmetric".

Name	Value	Description
direct	-6	Direct use of content encryption key (CEK)

Table 11: Direct Key

6.1.1.1. Security Considerations for Direct Key

This recipient algorithm has several potential problems that need to be considered:

* These keys need to have some method of being regularly updated over time. All of the content encryption algorithms specified in this document have limits on how many times a key can be used without significant loss of security.

* These keys need to be dedicated to a single algorithm. There have been a number of attacks developed over time when a single key is used for multiple different algorithms. One example of this is the use of a single key for both the CBC encryption mode and the CBC-MAC authentication mode.

* Breaking one message means all messages are broken. If an adversary succeeds in determining the key for a single message, then the key for all messages is also determined.

6.1.2. Direct Key with KDF

These recipient algorithms take a common shared secret between the two parties and apply the HKDF function (Section 5.1), using the context structure defined in Section 5.2 to transform the shared secret into the CEK. The "protected" field can be of nonzero length. Either the "salt" parameter for HKDF (Table 9) or the "PartyU nonce" parameter for the context structure (Table 10) MUST be present (both can be present if desired). The value in the "salt"/"nonce" parameter can be generated either randomly or deterministically. The requirement is that it be a unique value for the shared secret in question.

If the salt/nonce value is generated randomly, then it is suggested that the length of the random value be the same length as the output of the hash function underlying HKDF. While there is no way to guarantee that it will be unique, there is a high probability that it will be unique. If the salt/nonce value is generated deterministically, it can be guaranteed to be unique, and thus there

is no length requirement.

A new IV must be used for each message if the same key is used. The IV can be modified in a predictable manner, a random manner, or an unpredictable manner (e.g., encrypting a counter).

The IV used for a key can also be generated using the same HKDF functionality used to generate the key. If HKDF is used for generating the IV, the algorithm identifier is set to 34 ("IV-GENERATION").

The set of algorithms defined in this document can be found in Table 12.

Name	Value	KDF	Description
direct+HKDF-SHA-256	-10	HKDF SHA-256	Shared secret w/ HKDF and SHA-256
direct+HKDF-SHA-512	-11	HKDF SHA-512	Shared secret w/ HKDF and SHA-512
direct+HKDF-AES-128	-12	HKDF AES-MAC-128	Shared secret w/ AES-MAC 128-bit key
direct+HKDF-AES-256	-13	HKDF AES-MAC-256	Shared secret w/ AES-MAC 256-bit key

Table 12: Direct Key with KDF

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the algorithm being used.

* If the "key_ops" field is present, it MUST include "derive key" or "derive bits".

6.1.2.1. Security Considerations for Direct Key with KDF

The shared secret needs to have some method of being regularly updated over time. The shared secret forms the basis of trust. Although not used directly, it should still be subject to scheduled rotation.

These methods do not provide for perfect forward secrecy, as the same shared secret is used for all of the keys generated; however, if the key for any single message is discovered, only the message or series of messages using that derived key are compromised. A new key derivation step will generate a new key that requires the same amount of work to get the key.

6.2. Key Wrap

Key wrap is defined in Section 8.5.2 of [RFC9052]. Information about how to fill in the COSE_Recipient structure is detailed there.

6.2.1. AES Key Wrap

The AES Key Wrap algorithm is defined in [RFC3394]. This algorithm uses an AES key to wrap a value that is a multiple of 64 bits. As

such, it can be used to wrap a key for any of the content encryption algorithms defined in this document. The algorithm requires a single fixed parameter, the initial value. This is fixed to the value specified in Section 2.2.3.1 of [RFC3394]. There are no public key parameters that vary on a per-invocation basis. The protected header bucket MUST be empty.

Keys may be obtained from either a key structure or a recipient structure. Implementations that are encrypting or decrypting MUST validate that the key type, key length, and algorithm are correct and appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "Symmetric".

* If the "alg" field is present, it MUST match the AES Key Wrap algorithm being used.

* If the "key_ops" field is present, it MUST include "encrypt" or "wrap key" when encrypting.

* If the "key_ops" field is present, it MUST include "decrypt" or "unwrap key" when decrypting.

Name	Value	Key Size	Description
A128KW	-3	128	AES Key Wrap w/ 128-bit key
A192KW	-4	192	AES Key Wrap w/ 192-bit key
A256KW	-5	256	AES Key Wrap w/ 256-bit key

Table 13: AES Key Wrap Algorithm Values

6.2.1.1. Security Considerations for AES Key Wrap

The shared secret needs to have some method of being regularly updated over time. The shared secret is the basis of trust.

6.3. Direct Key Agreement

Direct Key Agreement is defined in Section 8.5.4 of [RFC9052]. Information about how to fill in the COSE_Recipient structure is detailed there.

6.3.1. Direct ECDH

The mathematics for ECDH can be found in [RFC6090]. In this document, the algorithm is extended to be used with the two curves defined in [RFC7748].

ECDH is parameterized by the following:

Curve Type/Curve: The curve selected controls not only the size of the shared secret, but the mathematics for computing the shared secret. The curve selected also controls how a point in the curve is represented and what happens for the identity points on the curve. In this specification, we allow for a number of different curves to be used. A set of curves is defined in Table 18.

The math used to obtain the computed secret is based on the curve selected and not on the ECDH algorithm. For this reason, a new

algorithm does not need to be defined for each of the curves.

Computed Secret to Shared Secret: Once the computed secret is known, the resulting value needs to be converted to a byte string to run the KDF. The x-coordinate is used for all of the curves defined in this document. For curves X25519 and X448, the resulting value is used directly, as it is a byte string of a known length. For the P-256, P-384, and P-521 curves, the x-coordinate is run through the Integer-to-Octet-String primitive (I2OSP) function defined in [RFC8017], using the same computation for n as is defined in Section 2.1.

Ephemeral-Static or Static-Static: The key agreement process may be done using either a static or an ephemeral key for the sender's side. When using ephemeral keys, the sender MUST generate a new ephemeral key for every key agreement operation. The ephemeral key is placed in the "ephemeral key" parameter and MUST be present for all algorithm identifiers that use ephemeral keys. When using static keys, the sender MUST either generate a new random value or create a unique value for use as a KDF input. For the KDFs used, this means that either the "salt" parameter for HKDF (Table 9) or the "PartyU nonce" parameter for the context structure (Table 10) MUST be present (both can be present if desired). The value in the parameter MUST be unique for the pair of keys being used. It is acceptable to use a global counter that is incremented for every Static-Static operation and use the resulting value. Care must be taken that the counter is saved to permanent storage in a way that avoids reuse of that counter value. When using static keys, the static key should be identified to the recipient. The static key can be identified by providing either the key ("static key") or a key identifier for the static key ("static key id"). Both of these header parameters are defined in Table 15.

Key Derivation Algorithm: The result of an ECDH key agreement process does not provide a uniformly random secret. As such, it needs to be run through a KDF in order to produce a usable key. Processing the secret through a KDF also allows for the introduction of context material: how the key is going to be used and one-time material for Static-Static key agreement. All of the algorithms defined in this document use one of the HKDF algorithms defined in Section 5.1 with the context structure defined in Section 5.2.

Key Wrap Algorithm: No key wrap algorithm is used. This is represented in Table 14 as "none". The key size for the context structure is the content layer encryption algorithm size.

COSE does not have an Ephemeral-Ephemeral version defined. The reason for this is that COSE is not an online protocol by itself and thus does not have a method of establishing ephemeral secrets on both sides. The expectation is that a protocol would establish the secrets for both sides, and then they would be used as Static-Static for the purposes of COSE, or that the protocol would generate a shared secret and a direct encryption would be used.

The set of direct ECDH algorithms defined in this document is found in Table 14.

Name	Value	KDF	Ephemeral-Static	Key Wrap	Description
ECDH-ES + HKDF-256	-25	HKDF -- SHA-256	yes	none	ECDH ES w/ HKDF -- generate key directly

ECDH-ES + HKDF-512	-26	HKDF -- SHA-512	yes	none	ECDH ES w/ HKDF -- generate key directly
ECDH-SS + HKDF-256	-27	HKDF -- SHA-256	no	none	ECDH SS w/ HKDF -- generate key directly
ECDH-SS + HKDF-512	-28	HKDF -- SHA-512	no	none	ECDH SS w/ HKDF -- generate key directly

Table 14: ECDH Algorithm Values

Name	Label	Type	Algorithm	Description
ephemeral key	-1	COSE_Key	ECDH-ES+HKDF-256, ECDH-ES+HKDF-512, ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW	Ephemeral public key for the sender
static key	-2	COSE_Key	ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	Static public key for the sender
static key id	-3	bstr	ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-SS+A128KW, ECDH-SS+A192KW, ECDH-SS+A256KW	Static public key identifier for the sender

Table 15: ECDH Algorithm Parameters

This document defines these algorithms to be used with the curves P-256, P-384, P-521, X25519, and X448. Implementations MUST verify that the key type and curve are correct. Different curves are restricted to different key types. Implementations MUST verify that the curve and algorithm are appropriate for the entities involved.

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "EC2" or "OKP".

* If the "alg" field is present, it MUST match the key agreement algorithm being used.

* If the "key_ops" field is present, it MUST include "derive key" or "derive bits" for the private key.

* If the "key_ops" field is present, it MUST be empty for the public key.

6.3.1.1. Security Considerations for ECDH

There is a method of checking that points provided from external

entities are valid. For the "EC2" key format, this can be done by checking that the x and y values form a point on the curve. For the "OKP" format, there is no simple way to perform point validation.

Consideration was given to requiring that the public keys of both entities be provided as part of the key derivation process (as recommended in Section 6.1 of [RFC7748]). This was not done, because COSE is used in a store-and-forward format rather than in online key exchange. In order for this to be a problem, either the receiver public key has to be chosen maliciously or the sender has to be malicious. In either case, all security evaporates anyway.

A proof of possession of the private key associated with the public key is recommended when a key is moved from untrusted to trusted (either by the end user or by the entity that is responsible for making trust statements on keys).

6.4. Key Agreement with Key Wrap

Key Agreement with Key Wrap is defined in Section 8.5.5 of [RFC9052]. Information about how to fill in the COSE_Recipient structure is detailed there.

6.4.1. ECDH with Key Wrap

These algorithms are defined in Table 16.

ECDH with Key Agreement is parameterized by the same header parameters as for ECDH; see Section 6.3.1, with the following modifications:

Key Wrap Algorithm: Any of the key wrap algorithms defined in Section 6.2 are supported. The size of the key used for the key wrap algorithm is fed into the KDF. The set of identifiers is found in Table 16.

Name	Value	KDF	Ephemeral-Static	Key Wrap	Description
ECDH-ES + A128KW	-29	HKDF -- SHA-256	yes	A128KW	ECDH ES w/ HKDF and AES Key Wrap w/ 128-bit key
ECDH-ES + A192KW	-30	HKDF -- SHA-256	yes	A192KW	ECDH ES w/ HKDF and AES Key Wrap w/ 192-bit key
ECDH-ES + A256KW	-31	HKDF -- SHA-256	yes	A256KW	ECDH ES w/ HKDF and AES Key Wrap w/ 256-bit key
ECDH-SS + A128KW	-32	HKDF -- SHA-256	no	A128KW	ECDH SS w/ HKDF and AES Key Wrap w/ 128-bit key
ECDH-SS + A192KW	-33	HKDF -- SHA-256	no	A192KW	ECDH SS w/ HKDF and AES Key Wrap w/ 192-bit key
ECDH-SS + A256KW	-34	HKDF -- SHA-256	no	A256KW	ECDH SS w/ HKDF and AES

					Key Wrap w/ 256-bit key

Table 16: ECDH Algorithm Values with Key Wrap

When using a COSE key for this algorithm, the following checks are made:

* The "kty" field MUST be present, and it MUST be "EC2" or "OKP".

* If the "alg" field is present, it MUST match the key agreement algorithm being used.

* If the "key_ops" field is present, it MUST include "derive key" or "derive bits" for the private key.

* If the "key_ops" field is present, it MUST be empty for the public key.

7. Key Object Parameters

The COSE_Key object defines a way to hold a single key object. It is still required that the members of individual key types be defined. This section of the document is where we define an initial set of members for specific key types.

For each of the key types, we define both public and private members. The public members are what is transmitted to others for their usage. Private members allow individuals to archive keys. However, there are some circumstances in which private keys may be distributed to entities in a protocol. Examples include: entities that have poor random number generation, centralized key creation for multicast-type operations, and protocols in which a shared secret is used as a bearer token for authorization purposes.

Key types are identified by the "kty" member of the COSE_Key object. In this document, we define four values for the member:

Name	Value	Description
OKP	1	Octet Key Pair
EC2	2	Elliptic Curve Keys w/ x- and y-coordinate pair
Symmetric	4	Symmetric Keys
Reserved	0	This value is reserved

Table 17: Key Type Values

7.1. Elliptic Curve Keys

Two different key structures are defined for elliptic curve keys. One version uses both an x-coordinate and a y-coordinate, potentially with point compression ("EC2"). This is the conventional elliptic curve (EC) point representation that is used in [RFC5480]. The other version uses only the x-coordinate, as the y-coordinate is either to be recomputed or not needed for the key agreement operation ("OKP").

Applications MUST check that the curve and the key type are consistent and reject a key if they are not.

Name	Value	Key Type	Description
P-256	1	EC2	NIST P-256, also known as secp256r1
P-384	2	EC2	NIST P-384, also known as secp384r1
P-521	3	EC2	NIST P-521, also known as secp521r1
X25519	4	OKP	X25519 for use w/ ECDH only
X448	5	OKP	X448 for use w/ ECDH only
Ed25519	6	OKP	Ed25519 for use w/ EdDSA only
Ed448	7	OKP	Ed448 for use w/ EdDSA only

Table 18: Elliptic Curves

7.1.1. Double Coordinate Curves

Generally, protocols transmit elliptic-curve points as either the x-coordinate and y-coordinate or the x-coordinate and a sign bit for the y-coordinate. The latter encoding has not been recommended by the IETF due to potential IPR issues. However, for operations in constrained environments, the ability to shrink a message by not sending the y-coordinate is potentially useful.

For EC keys with both coordinates, the "kty" member is set to 2 (EC2). The key parameters defined in this section are summarized in Table 19. The members that are defined for this key type are:

crv: This contains an identifier of the curve to be used with the key. The curves defined in this document for this key type can be found in Table 18. Other curves may be registered in the future, and private curves can be used as well.

x: This contains the x-coordinate for the EC point. The integer is converted to a byte string as defined in [SEC1]. Leading-zero octets MUST be preserved.

y: This contains either the sign bit or the value of the y-coordinate for the EC point. When encoding the value y, the integer is converted to a byte string (as defined in [SEC1]) and encoded as a CBOR bstr. Leading-zero octets MUST be preserved. Compressed point encoding is also supported. Compute the sign bit as laid out in the Elliptic-Curve-Point-to-Octet-String Conversion function of [SEC1]. If the sign bit is zero, then encode y as a CBOR false value; otherwise, encode y as a CBOR true value. The encoding of the infinity point is not supported.

d: This contains the private key.

For public keys, it is REQUIRED that "crv", "x", and "y" be present in the structure. For private keys, it is REQUIRED that "crv" and "d" be present in the structure. For private keys, it is RECOMMENDED that "x" and "y" also be present, but they can be recomputed from the required elements, and omitting them saves on space.

Key Type	Name	Label	CBOR Type	Description
2	crv	-1	int /	EC identifier -- Taken from the

			tstr	"COSE Elliptic Curves" registry
2	x	-2	bstr	x-coordinate
2	y	-3	bstr / bool	y-coordinate
2	d	-4	bstr	Private key

Table 19: EC Key Parameters

7.2. Octet Key Pair

A new key type is defined for Octet Key Pairs (OKPs). Do not assume that keys using this type are elliptic curves. This key type could be used for other curve types (for example, mathematics based on hyper-elliptic surfaces).

The key parameters defined in this section are summarized in Table 20. The members that are defined for this key type are:

crv: This contains an identifier of the curve to be used with the key. The curves defined in this document for this key type can be found in Table 18. Other curves may be registered in the future, and private curves can be used as well.

x: This contains the public key. The byte string contains the public key as defined by the algorithm. (For X25519, internally it is a little-endian integer.)

d: This contains the private key.

For public keys, it is REQUIRED that "crv" and "x" be present in the structure. For private keys, it is REQUIRED that "crv" and "d" be present in the structure. For private keys, it is RECOMMENDED that "x" also be present, but it can be recomputed from the required elements, and omitting it saves on space.

Name	Key Type	Label	Type	Description
crv	1	-1	int / tstr	EC identifier -- Taken from the "COSE Elliptic Curves" registry
x	1	-2	bstr	Public Key
d	1	-4	bstr	Private key

Table 20: Octet Key Pair Parameters

7.3. Symmetric Keys

Occasionally, it is required that a symmetric key be transported between entities. This key structure allows for that to happen.

For symmetric keys, the "kty" member is set to 4 ("Symmetric"). The member that is defined for this key type is:

k: This contains the value of the key.

This key structure does not have a form that contains only public members. As it is expected that this key structure is going to be transmitted, care must be taken that it is never transmitted

accidentally or insecurely. For symmetric keys, it is REQUIRED that "k" be present in the structure.

Name	Key Type	Label	Type	Description
k	4	-1	bstr	Key Value

Table 21: Symmetric Key Parameters

8. COSE Capabilities

The capabilities of an algorithm or key type need to be specified in some situations. This has a counterpart in the S/MIME specifications, where SMIMECapabilities is defined in Section 2.5.2 of [RFC8551]. This document defines the same concept for COSE.

The algorithm identifier is not included in the capabilities data, as it should be encoded elsewhere in the message. The key type identifier is included in the capabilities data, as it is not expected to be encoded elsewhere.

Two different types of capabilities are defined: capabilities for algorithms and capabilities for key type. Once defined by registration with IANA, the list of capabilities for an algorithm or key type is immutable. If it is later found that a new capability is needed for a key type or algorithm, it will require that a new code point be assigned to deal with that. As a general rule, the capabilities are going to map to algorithm-specific header parameters or key parameters, but they do not need to do so. An example of this is the HSS-LMS key type capabilities defined below, where the hash algorithm used is included.

The capability structure is an array of values; the values included in the structure are dependent on a specific algorithm or key type. For algorithm capabilities, the first element should always be a key type value if applicable, but the items that are specific to a key (for example, a curve) should not be included in the algorithm capabilities. This means that if one wishes to enumerate all of the capabilities for a device that implements ECDH, it requires that all of the combinations of algorithms and key pairs be specified. The last example of Section 8.3 provides a case where this is done by allowing for a cross product to be specified between an array of algorithm capabilities and key type capabilities (see the ECDH-ES+A25KW element). For a key, the first element should be the key type value. While this means that the key type value will be duplicated if both an algorithm and key capability are used, the key type is needed in order to understand the rest of the values.

8.1. Assignments for Existing Algorithms

For the current set of algorithms in the registry other than IV-GENERATION (those in this document as well as those in [RFC8230], [RFC8778], and [RFC9021]), the capabilities list is an array with one element, the key type (from the "COSE Key Types" Registry). It is expected that future registered algorithms could have zero, one, or multiple elements.

8.2. Assignments for Existing Key Types

There are a number of pre-existing key types; the following deals with creating the capability definition for those structures:

* OKP, EC2: The list of capabilities is:

- The key type value. (1 for OKP or 2 for EC2.)

- One curve for that key type from the "COSE Elliptic Curves" registry.

* RSA: The list of capabilities is:

  - The key type value (3).

* Symmetric: The list of capabilities is:

  - The key type value (4).

* HSS-LMS: The list of capabilities is:

  - The key type value (5).

  - Algorithm identifier for the underlying hash function from the "COSE Algorithms" registry.

* WalnutDSA: The list of capabilities is:

  - The key type value (6).

  - The N value (group and matrix size) for the key, a uint.

  - The q value (finite field order) for the key, a uint.

8.3. Examples

Capabilities can be used in a key derivation process to make sure that both sides are using the same parameters. The three examples below show different ways that one might utilize parameters in specifying an application protocol:

* Only an algorithm capability: This is useful if the protocol wants to require a specific algorithm, such as ES256, but it is agnostic about which curve is being used. This requires that the algorithm identifier be specified in the protocol. See the first example.

* Only a key type capability: This is useful if the protocol wants to require a specific key type and curve, such as P-256, but will accept any algorithm using that curve (e.g., both ECDSA and ECDH). See the second example.

* Both algorithm and key type capabilities: This is used if the protocol needs to nail down all of the options surrounding an algorithm -- e.g., EdDSA with the curve Ed25519. As with the first example, the algorithm identifier needs to be specified in the protocol. See the third example, which just concatenates the two capabilities together.

Algorithm ES256

```
0x8102                  / [2 \ EC2 \ ] /
```

Key type EC2 with P-256 curve:

```
0x820201                / [2 \ EC2 \, 1 \ P-256 \] /
```

ECDH-ES + A256KW with an X25519 curve:

```
0x8101820104            / [1 \ OKP \],[1 \ OKP \, 4 \ X25519 \] /
```

The capabilities can also be used by an entity to advertise what it is capable of doing. The decoded example below is one of many

encodings that could be used for that purpose. Each array element includes three fields: the algorithm identifier, one or more algorithm capabilities, and one or more key type capabilities.

```
[
 [-8 / EdDSA /,
   [1 / OKP key type /],
   [
     [1 / OKP /, 6 / Ed25519 / ],
     [1 /OKP/, 7 /Ed448 /]
   ]
 ],
 [-7 / ECDSA with SHA-256/,
   [2 /EC2 key type/],
   [
     [2 /EC2/, 1 /P-256/],
     [2 /EC2/, 3 /P-521/]
   ]
 ],
 [ -31 / ECDH-ES+A256KW/,
   [
     [ 2 /EC2/],
     [1 /OKP/ ]
   ],
   [
     [2 /EC2/, 1 /P-256/],
     [1 /OKP/, 4 / X25519/ ]
   ]
 ],
 [ 1 / A128GCM /,
   [ 4 / Symmetric / ],
   [ 4 / Symmetric /]
 ]
]
```

Examining the above:

* The first element indicates that the entity supports EdDSA with curves Ed25519 and Ed448.

* The second element indicates that the entity supports ECDSA with SHA-256 with curves P-256 and P-521.

* The third element indicates that the entity supports Ephemeral-Static ECDH using AES256 key wrap. The entity can support the P-256 curve with an EC2 key type and the X25519 curve with an OKP key type.

* The last element indicates that the entity supports AES-GCM of 128 bits for content encryption.

The entity does not advertise that it supports any MAC algorithms.

9. CBOR Encoding Restrictions

This document limits the restrictions it imposes on how the CBOR Encoder needs to work. The new encoding restrictions are aligned with the Core Deterministic Encoding Requirements specified in Section 4.2.1 of RFC 8949 [STD94]. It has been narrowed down to the following restrictions:

* The restriction applies to the encoding of the COSE_KDF_Context.

* Encoding MUST be done using definite lengths, and the length of the (encoded) argument MUST be the minimum possible length. This means that the integer 1 is encoded as "0x01" and not "0x1801".

* Applications MUST NOT generate messages with the same label used twice as a key in a single map. Applications MUST NOT parse and process messages with the same label used twice as a key in a single map. Applications can enforce the parse-and-process requirement by using parsers that will fail the parse step or by using parsers that will pass all keys to the application, and the application can perform the check for duplicate keys.

10. IANA Considerations

IANA has updated all COSE registries except for "COSE Header Parameters" and "COSE Key Common Parameters" to point to this document instead of [RFC8152].

10.1. Changes to the "COSE Key Types" Registry

IANA has added a new column in the "COSE Key Types" registry. The new column is labeled "Capabilities" and has been populated according to the entries in Table 22.

Value	Name	Capabilities
1	OKP	[kty(1), crv]
2	EC2	[kty(2), crv]
3	RSA	[kty(3)]
4	Symmetric	[kty(4)]
5	HSS-LMS	[kty(5), hash algorithm]
6	WalnutDSA	[kty(6), N value, q value]

Table 22: Key Type Capabilities

10.2. Changes to the "COSE Algorithms" Registry

IANA has added a new column in the "COSE Algorithms" registry. The new column is labeled "Capabilities" and has been populated with "[kty]" for all current, nonprovisional registrations.

IANA has updated the Reference column in the "COSE Algorithms" registry to include this document as a reference for all rows where it was not already present.

IANA has added a new row to the "COSE Algorithms" registry.

Name	Value	Description	Reference	Recommended
IV-GENERATION	34	For doing IV generation for symmetric algorithms.	RFC 9053	No

Table 23: New entry in the COSE Algorithms registry

The Capabilities column for this registration is to be empty.

10.3. Changes to the "COSE Key Type Parameters" Registry

IANA has modified the description to "Public Key" for the line with "Key Type" of 1 and the "Name" of "x". See Table 20, which has been modified with this change.

10.4. Expert Review Instructions

All of the IANA registries established by [RFC8152] are, at least in part, defined as Expert Review [RFC8126]. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take the following into consideration:

* Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate an existing registration and that the code point is likely to be used in deployments. The ranges tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.

* Standards Track or BCP RFCs are required to register a code point in the Standards Action range. Specifications should exist for Specification Required ranges, but early assignment before an RFC is available is considered to be permissible. Specifications are needed for the first-come, first-served range if the points are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

* Experts should take into account the expected usage of fields when approving code point assignment. The fact that the Standards Action range is only available to Standards Track documents does not mean that a Standards Track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left and the size of device it will be used on.

* When algorithms are registered, vanity registrations should be discouraged. One way to do this is to require registrations to provide additional documentation on security analysis of the algorithm. Another thing that should be considered is requesting an opinion on the algorithm from the Crypto Forum Research Group (CFRG). Algorithms are expected to meet the security requirements of the community and the requirements of the message structures in order to be suitable for registration.

11. Security Considerations

There are a number of security considerations that need to be taken into account by implementers of this specification. The security considerations that are specific to an individual algorithm are placed next to the description of the algorithm. While some considerations have been highlighted here, additional considerations may be found in the documents listed in the references.

Implementations need to protect the private key material for all individuals. Some cases in this document need to be highlighted with regard to this issue.

* Use of the same key for two different algorithms can leak information about the key. It is therefore recommended that keys be restricted to a single algorithm.

* Use of "direct" as a recipient algorithm combined with a second recipient algorithm exposes the direct key to the second recipient; Section 8.5 of [RFC9052] forbids combining "direct" recipient algorithms with other modes.

* Several of the algorithms in this document have limits on the number of times that a key can be used without leaking information about the key.

The use of ECDH and direct plus KDF (with no key wrap) will not directly lead to the private key being leaked; the one-way function of the KDF will prevent that. There is, however, a different issue that needs to be addressed. Having two recipients requires that the CEK be shared between two recipients. The second recipient therefore has a CEK that was derived from material that can be used for the weak proof of origin. The second recipient could create a message using the same CEK and send it to the first recipient; the first recipient would, for either Static-Static ECDH or direct plus KDF, make an assumption that the CEK could be used for proof of origin, even though it is from the wrong entity. If the key wrap step is added, then no proof of origin is implied and this is not an issue.

Although it has been mentioned before, it bears repeating that the use of a single key for multiple algorithms has been demonstrated in some cases to leak information about a key, providing the opportunity for attackers to forge integrity tags or gain information about encrypted content. Binding a key to a single algorithm prevents these problems. Key creators and key consumers are strongly encouraged to not only create new keys for each different algorithm, but to include that selection of algorithm in any distribution of key material and strictly enforce the matching of algorithms in the key structure to algorithms in the message structure. In addition to checking that algorithms are correct, the key form needs to be checked as well. Do not use an "EC2" key where an "OKP" key is expected.

Before using a key for transmission, or before acting on information received, a trust decision on a key needs to be made. Is the data or action something that the entity associated with the key has a right to see or a right to request? A number of factors are associated with this trust decision. Some highlighted here are:

* What are the permissions associated with the key owner?

* Is the cryptographic algorithm acceptable in the current context?

* Have the restrictions associated with the key, such as algorithm or freshness, been checked, and are they correct?

* Is the request something that is reasonable, given the current state of the application?

* Have any security considerations that are part of the message been enforced (as specified by the application or "crit" header parameter)?

There are a large number of algorithms presented in this document that use nonce values. For all of the nonces defined in this document, there is some type of restriction on the nonce being a unique value for either a key or some other conditions. In all of these cases, there is no known requirement on the nonce being both unique and unpredictable; under these circumstances, it’s reasonable to use a counter for creation of the nonce. In cases where one wants the pattern of the nonce to be unpredictable as well as unique, one can use a key created for that purpose and encrypt the counter to produce the nonce value.

One area that has been getting exposure is traffic analysis of encrypted messages based on the length of the message. This specification does not provide a uniform method for providing padding as part of the message structure. An observer can distinguish between two different messages (for example, "YES" and "NO") based on the length for all of the content encryption algorithms that are defined in this document. This means that it is up to the applications to document how content padding is to be done in order to prevent or discourage such analysis. (For example, the text strings could be defined as "YES" and "NO ".)

The analysis done in [RFC9147] is based on the number of records that are sent. This should map well to the number of messages sent when using COSE, so that analysis should hold here as well, under the assumption that the COSE messages are roughly the same size as DTLS records. It needs to be noted that the limits are based on the number of messages, but QUIC and DTLS are always pairwise-based endpoints. In contrast, [OSCORE-GROUPCOMM] uses COSE in a group communication scenario. Under these circumstances, it may be that no one single entity will see all of the messages that are encrypted, and therefore no single entity can trigger the rekey operation.

12. References

12.1. Normative References

[AES-GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, DOI 10.6028/NIST.SP.800-38D, November 2007, <https://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>.

[DSS] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[MAC] Menezes, A., van Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, Boca Raton, 1996, <https://cacr.uwaterloo.ca/hac/>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/info/rfc3394>.

[RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <https://www.rfc-editor.org/info/rfc3610>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[RFC8439] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 8439, DOI 10.17487/RFC8439, June 2018, <https://www.rfc-editor.org/info/rfc8439>.

[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>.

[SEC1] Certicom Research, "SEC 1: Elliptic Curve Cryptography", Standards for Efficient Cryptography, May 2009, <https://www.secg.org/sec1-v2.pdf>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, December 2020, <https://www.rfc-editor.org/info/std94>.

12.2. Informative References

[CFRG-DET-SIGS]
Mattsson, J. P., Thormarker, E., and S. Ruohomaa, "Deterministic ECDSA and EdDSA Signatures with Additional Randomness", Work in Progress, Internet-Draft, draft-mattsson-cfrg-det-sigs-with-noise-04, 15 February 2022, <https://datatracker.ietf.org/doc/html/draft-mattsson-cfrg-det-sigs-with-noise-04>.

[COUNTERSIGN]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-08, 22 August 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-countersign-08>.

[GitHub-Examples]
"GitHub Examples of COSE", commit 3221310, 3 June 2020, <https://github.com/cose-wg/Examples>.

[HKDF] Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", 2010, <https://eprint.iacr.org/2010/264.pdf>.

[OSCORE-GROUPCOMM] Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-14, 7 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-14>.

[RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, DOI 10.17487/RFC4231, December 2005, <https://www.rfc-editor.org/info/rfc4231>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

[RFC8230] Jones, M., "Using RSA Algorithms with CBOR Object Signing and Encryption (COSE) Messages", RFC 8230, DOI 10.17487/RFC8230, September 2017, <https://www.rfc-editor.org/info/rfc8230>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8778] Housley, R., "Use of the HSS/LMS Hash-Based Signature Algorithm with CBOR Object Signing and Encryption (COSE)", RFC 8778, DOI 10.17487/RFC8778, April 2020, <https://www.rfc-editor.org/info/rfc8778>.

[RFC9021] Atkins, D., "Use of the Walnut Digital Signature Algorithm with CBOR Object Signing and Encryption (COSE)", RFC 9021, DOI 10.17487/RFC9021, May 2021, <https://www.rfc-editor.org/info/rfc9021>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[ROBUST] Fischlin, M., Gnther, F., and C. Janson, "Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS", February 2020, <https://eprint.iacr.org/2020/718.pdf>.

[SP800-38D] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>.

[SP800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A, Revision 3, DOI 10.6028/NIST.SP.800-56Ar3, April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf>.

[STD90] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017, <https://www.rfc-editor.org/info/std90>.

Acknowledgments

This document is a product of the COSE Working Group of the IETF.

The following individuals are to blame for getting me started on this project in the first place: Richard Barnes, Matt Miller, and Martin Thomson.

The initial draft version of the specification was based to some degree on the outputs of the JOSE and S/MIME Working Groups.

The following individuals provided input into the final form of the document: Carsten Bormann, John Bradley, Brian Campbell, Michael B. Jones, Ilari Liusvaara, Francesca Palombini, Ludwig Seitz, and Gran Selander.

Author’s Address

Jim Schaad

August Cellars