CBOR Object Signing and Encryption (COSE): Initial Algorithms
draft-ietf-cose-rfc8152bis-algs-04

Abstract

   Concise Binary Object Representation (CBOR) is a data format designed
   for small code size and small message size. There is a need for the
   ability to have basic security services defined for this data format.
   This document defines the CBOR Object Signing and Encryption (COSE)
   protocol. This specification describes how to create and process
   signatures, message authentication codes, and encryption using CBOR
   for serialization. COSE additionally describes how to represent
   cryptographic keys using CBOR.

   In this specification the conventions for the use of a number of
   cryptographic algorithms with COSE. The details of the structure of
   COSE are defined in [I-D.ietf-cose-rfc8152bis-struct].

   This document along with [I-D.ietf-cose-rfc8152bis-struct] obsoletes
   RFC8152.

Contributing to this document

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

   The source for this draft is being maintained in GitHub. Suggested
   changes should be submitted as pull requests at https://github.com/
   cose-wg/cose-rfc8152bis. 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

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.

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)" [RFC7049]. CBOR extended the data model of the JavaScript
   Object Notation (JSON) [RFC8259] by allowing for binary data, among
   other changes. CBOR is being adopted by several of the IETF working
   groups dealing with the IoT world as their encoding of data
   structures. CBOR was designed specifically to be both small in terms
   of messages transport and implementation size and be 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.
   [I-D.ietf-cose-rfc8152bis-struct] contains the serialization
   structures and the procedures for using the different cryptographic
   algorithms. This document provides for an initial set of algorithms
   that are then use with those structures. Additional algorithms
   beyond what are in this document are defined elsewhere.

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 RFC8152

* Extract the sections dealing with specific algorithms into this
   document. The sections dealing with structure and general
   processing rules are placed in [I-D.ietf-cose-rfc8152bis-struct].

1.3. Document Terminology

In this document, we use the following terminology:

Byte is a synonym for octet.

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

Authenticated Encryption (AE) [RFC5116] algorithms are those
encryption algorithms that provide an authentication check of the
plain text contents as part of the encryption service.

Authenticated Encryption with Associated Data (AEAD) [RFC5116]
algorithms provide the same content authentication service as AE
algorithms, but they additionally provide for authentication of non-
encrypted data as well.

1.4. CBOR Grammar

At the time that [RFC8152] was initially published, the CBOR Data
Definition Language (CDDL) [RFC8610] had not yet been published.
This document uses a variant of CDDL which is described in
[I-D.ietf-cose-rfc8152bis-struct]

1.5. Examples

A GitHub project has been created at <https://github.com/cose-wg/
Examples> that contains a set of testing examples as well. 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
in debugging the example and the output of the example presented in
both a hex and a CBOR diagnostic notation format. Some of the
examples at the site are designed failure testing cases; these are

clearly marked as such in the JSON file. If errors in the examples
in this document are found, the examples on GitHub will be updated,
and a note to that effect will be placed in the JSON file.

2. Signature Algorithms

Section 9.1 of [I-D.ietf-cose-rfc8152bis-struct]
[I-D.ietf-cose-rfc8152bis-struct] contains a generic description of
signature algorithms. The document defines signature algorithm
identifiers for two signature algorithms.

2.1. ECDSA

ECDSA [DSS] defines a signature algorithm using ECC. Implementations
SHOULD use a deterministic version of ECDSA such as the one defined
in [RFC6979]. The use of a deterministic signature algorithm allows
for 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 leads to leaked keys. It additionally
allows for doing 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 to work only with the curves P-256,
P-384, and P-521. This document requires that the curves be encoded
using the 'EC2' (2 coordinate elliptic curve) key type.
Implementations need to check that the key type and curve are correct

   when creating and verifying a signature. Other documents can define
   it to work with other curves and points 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 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 bit 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

   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 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 requirement on the new curve is
   that its order be the same as the old one and 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.

* Change 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 down, 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.

2.2. Edwards-Curve Digital Signature Algorithms (EdDSAs)

[RFC8032] describes the elliptic curve signature scheme Edwards-curve
Digital Signature Algorithm (EdDSA). In that document, the signature
algorithm is instantiated using parameters for 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 pre-hash 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. This means that
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 algorithms 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 Values

[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

How public values are computed is not the same when looking at EdDSA
and Elliptic Curve Diffie-Hellman (ECDH); for this reason, they
should not be used with the other algorithm.

If batch signature verification is performed, a well-seeded
cryptographic random number generator is REQUIRED. Signing and non-
batch signature verification are deterministic operations and do not
need random numbers of any kind.

3. Message Authentication Code (MAC) Algorithms

Section 9.2 of [I-D.ietf-cose-rfc8152bis-struct]
[I-D.ietf-cose-rfc8152bis-struct] 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 algorithm was also designed to allow for new hash
   algorithms to be directly plugged in without changes to the hash
   function. The HMAC design process has been shown as solid since,
   while the security of hash algorithms 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 algorithms 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 | 4 | SHA-256 | 64 | HMAC w/ SHA-256 |
   | 256/64 | | | | truncated to 64 bits |
   +-------------+-------+---------+------------+----------------------+
   | HMAC | 5 | SHA-256 | 256 | HMAC w/ SHA-256 |
   | 256/256 | | | | |
   +-------------+-------+---------+------------+----------------------+
   | HMAC | 6 | SHA-384 | 384 | HMAC w/ SHA-384 |
   | 384/384 | | | | |
   +-------------+-------+---------+------------+----------------------+
   | HMAC | 7 | SHA-512 | 512 | HMAC w/ SHA-512 |
   | 512/512 | | | | |
   +-------------+-------+---------+------------+----------------------+

                      Table 3: HMAC Algorithm Values

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

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 defined in [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 IV used. For all of these algorithms, the IV is
fixed to all zeros. We provide an array of algorithms for various
key lengths 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 either from a key structure or from 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

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.)

* 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 9.3 of [[I-D.ietf-cose-rfc8152bis-struct](#)]
[[I-D.ietf-cose-rfc8152bis-struct](#)] 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 authenticated
encryption 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 are 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 Value for AES-GCM

Keys may be obtained either from a key structure or from a recipient
structure. Implementations encrypting and 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

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

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

* The total amount of data encrypted for a single key MUST NOT
   exceed $2^{39} - 256$ bits. An explicit check is required only in
   environments where it is expected that it might be exceeded.

Consideration was given to supporting smaller tag values; the
constrained community would desire tag sizes in the 64-bit range.
Doing so drastically changes both the maximum messages 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 a commonly used content encryption algorithm 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. 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	k	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 either from a key structure or from a recipient
structure. Implementations encrypting and 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

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 limitation is the sum of times the
   block cipher is used in computing the MAC value and in performing
   stream encryption operations. An explicit check is required only
   in environments where it is expected that it might be exceeded.

[RFC3610] additionally calls out one other consideration of note. It
is possible to do a pre-computation 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 to be 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 a 256-bit key and a 96-bit nonce, as well
as the plaintext and additional data as inputs and produces the
ciphertext as an option. 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 AES-GCM

Keys may be obtained either from a key structure or from a recipient
structure. Implementations encrypting and 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

The key and nounce 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.

5. Key Derivation Functions (KDFs)

   Section 9.4 of [I-D.ietf-cose-rfc8152bis-struct]
   [I-D.ietf-cose-rfc8152bis-struct] 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, and
   6.4.

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

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

   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 amount of salt 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-    | AES-CBC-MAC-128   | HKDF using AES-MAC as |
    | MAC-128      |                   | the PRF w/ 128-bit key |
    +--------------+-------------------+-----------------------+
    | HKDF AES-    | AES-CBC-MAC-256   | HKDF using AES-MAC as |
    | MAC-256      |                   | the PRF w/ 256-bit key |
    +--------------+-------------------+-----------------------+
```

Table 8: HKDF Algorithms

```
    +------+-------+------+--------------------------+-------------+
    | Name | Label | Type | Algorithm                | Description |
    +======+=======+======+==========================+=============+
    | salt | -20   | bstr | direct+HKDF-SHA-256,      | Random salt |
    |      |       |      | 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 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 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 doing 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 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 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 already are
      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,	Party U identity information

			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 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	Party U 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	Party U 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,	Party V identity information	

```
|          |       |      | ECDH-ES+A128KW,           |             |
|          |       |      | ECDH-ES+A192KW,           |             |
|          |       |      | ECDH-ES+A256KW,           |             |
|          |       |      | ECDH-SS+A128KW,           |             |
|          |       |      | ECDH-SS+A192KW,           |             |
|          |       |      | ECDH-SS+A256KW            |             |
+----------+-------+------+---------------------------+-------------+
| PartyV   | -25   | bstr | direct+HKDF-SHA-256,      | Party V     |
| nonce    |       | /    | direct+HKDF-SHA-512,      | provided    |
|          |       | int  | direct+HKDF-AES-128,      | nonce       |
|          |       |      | 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   | -26   | bstr | direct+HKDF-SHA-256,      | Party V     |
| other    |       |      | direct+HKDF-SHA-512,      | other       |
|          |       |      | direct+HKDF-AES-128,      | provided    |
|          |       |      | direct+HKDF-AES-256,      | information |
|          |       |      | 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 if the same environment is used for different
algorithms, then completely different keys will be
generated for each of those algorithms. This practice
means if algorithm A is broken and thus is easier to
find, the key derived for algorithm B will not be the
same as the key derived for algorithm A.

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

identity: This contains the identity information for
party U. 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 they send or
receive. The second way for a protocol to
assign identities is to use a name based on
a naming system (i.e., DNS, 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
either be implicit from the protocol or be
carried as a value in the unprotected
headers.

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

                              the value determined from elsewhere.

                              This option does not need to be specified
                              and is set to nil in that case.

                  other: This contains other information that is
                             defined by the protocol. This option does
                             not need to be specified and is set to nil
                             in that case.

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

      SuppPubInfo: This field contains public information that is
                    mutually known to both parties.

                  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 longer key size will not
                                  contain the key for 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. An
                               example is that an application could
                               define two different 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 preexisting 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 9.5 of [I-D.ietf-cose-rfc8152bis-struct]
   [I-D.ietf-cose-rfc8152bis-struct] 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

   Direct encryption algorithm is defined in Section 9.5.1 of
   [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-struct].
   Information about how to fill in the COSE_Recipient structure are
   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 CEK |
+--------+-------+-------------------+
```

Table 11: Direct Key

6.1.1.1. Security Considerations

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

* These keys need to have some method to be 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 applies 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 non-zero
length. Either the 'salt' parameter of HKDF or the 'PartyU nonce'
parameter of the context structure MUST be present. 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 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 (i.e., encrypting a counter).

The IV used for a key can also be generated from the same HKDF functionality as the key is generated. If HKDF is used for generating the IV, the algorithm identifier is set to "IV-GENERATION".

When these algorithms are used, the key type MUST be 'symmetric'.

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- | Shared secret w/ |
| | | MAC-128 | AES-MAC 128-bit key |
+---------------------+-------+-------------+--------------------+
| direct+HKDF-AES-256 | -13 | HKDF AES- | Shared secret w/ |
| | | MAC-256 | 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 'deriveKey' or 'deriveBits'.

6.1.2.1. Security Considerations

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

   While these methods do not provide for perfect forward secrecy, as
   the same shared secret is used for all of the keys generated, 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. 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
   parameters that vary on a per-invocation basis. The protected header
   field MUST be empty.

   Keys may be obtained either from a key structure or from a recipient
   structure. Implementations encrypting and 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. Security Considerations for AES-KW

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

6.3. 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 are 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 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 the KDFs used, this means 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. When using static keys,
the static key should be identified to the recipient. The static
key can be identified either by providing the key ('static key')
or by providing a key identifier for the static key ('static key
id'). Both of these 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 an online protocol by itself
and thus does not have a method to establish 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 are 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. Security Considerations

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 do 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 as 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. ECDH with Key Wrap

These algorithms are defined in Table 16.

ECDH with Key Agreement is parameterized by the same parameters as
for ECDH; see Section 6.3, 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 are
  found in Table 16.

Name	Value	KDF	Ephemeral-Static	Key Wrap	Description
===================	=======	====================	==================	==========	==
ECDH-ES + A128KW	-29	HKDF - SHA-256	yes	A128KW	ECDH ES w/ Concat KDF and AES Key Wrap w/ 128-bit key
ECDH-ES + A192KW	-30	HKDF - SHA-256	yes	A192KW	ECDH ES w/ Concat KDF and AES Key Wrap w/ 192-bit key
ECDH-ES + A256KW	-31	HKDF - SHA-256	yes	A256KW	ECDH ES w/ Concat KDF and AES Key Wrap w/ 256-bit key
ECDH-SS + A128KW	-32	HKDF - SHA-256	no	A128KW	ECDH SS w/ Concat KDF and AES Key Wrap w/ 128-bit key
ECDH-SS + A192KW	-33	HKDF - SHA-256	no	A192KW	ECDH SS w/ Concat KDF and AES Key Wrap w/ 192-bit key
ECDH-SS + A256KW	-34	HKDF - SHA-256	no	A256KW	ECDH SS w/ Concat KDF 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 for the archival of keys by individuals.
   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
   multi-cast 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 traditional 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

The traditional way of sending ECs has been to send either both 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 in
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 an octet 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 an octet string (as defined in [SEC1])
     and encoded as a CBOR bstr. Leading zero octets MUST be
     preserved. The 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	Name	Label	CBOR	Description
Type			Type	
======	======	=======	========	================================
2	crv	-1	int /	EC identifier - Taken from the
			tstr	"COSE Elliptic Curves" registry
2	x	-2	bstr	x-coordinate
2	y	-3	bstr /	y-coordinate
			bool	
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 (OKP). 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 x-coordinate for the EC point. The octet
         string represents a little-endian encoding of x.

   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 | Label | Type | Description |
   | | Type | | | |
   +======+==========+=======+=======+================================+
   | crv | 1 | -1 | int / | EC identifier - Taken from the |
   | | | | tstr | "COSE Elliptic Curves" registry|
   +------+----------+-------+-------+--------------------------------+
   | x | 1 | -2 | bstr | x-coordinate |
   +------+----------+-------+-------+--------------------------------+
   | 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. CBOR Encoding Restrictions

There has been an attempt to limit the number of places where the
document needs to impose restrictions on how the CBOR Encoder needs
to work. We have managed to narrow it 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
   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.

9. IANA Considerations

There are no IANA actions. The required actions are in
[I-D.ietf-cose-rfc8152bis-struct].

10. 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 any
individuals. There are some cases in this document that need to be
highlighted on this issue.

* Using 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.

* 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, the use of a single key for
multiple algorithms has been demonstrated in some cases to leak
information about a key, provide 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 not only to 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 of the ones that are 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' 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 either for a key or for 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 starting to get exposure is doing traffic
   analysis of encrypted messages based on the length of the message.
   This specification does not provide for a uniform method of providing
   padding as part of the message structure. An observer can
   distinguish between two different strings (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
   strings could be defined as 'YES' and 'NO '.)

11. References

11.1. Normative References

[AES-GCM] National Institute of Standards and Technology,
           "Recommendation for Block Cipher Modes of Operation:
           Galois/Counter Mode (GCM) and GMAC",
           DOI 10.6028/NIST.SP.800-38D, NIST Special
           Publication 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)", DOI 10.6028/NIST.FIPS.186-4,
           FIPS PUB 186-4, July 2013,
           <http://nvlpubs.nist.gov/nistpubs/FIPS/
           NIST.FIPS.186-4.pdf>.

[I-D.ietf-cose-rfc8152bis-struct]
           Schaad, J., "CBOR Object Signing and Encryption (COSE):
           Structures and Process", draft-ietf-cose-rfc8152bis-
           struct-05 (work in progress), August 18, 2019,
           <https://www.ietf.org/archive/id/draft-ietf-cose-
           rfc8152bis-struct-05>.

[MAC]   National Institute of Standards and Technology, "Computer
           Data Authentication", FIPS PUB 113, May 1985,
           <http://csrc.nist.gov/publications/fips/fips113/
           fips113.html>.

[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>.

[RFC7049]   Bormann, C. and P. Hoffman, "Concise Binary Object
            Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
            October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[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>.

[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>.

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

11.2. Informative References

[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>.

   [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>.

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

   [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
              Interchange Format", STD 90, RFC 8259,
              DOI 10.17487/RFC8259, December 2017,
              <https://www.rfc-editor.org/info/rfc8259>.

   [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>.

   [SP800-56A]
              Barker, E., Chen, L., Roginsky, A., and M. Smid,
              "Recommendation for Pair-Wise Key Establishment Schemes
              Using Discrete Logarithm Cryptography",
              DOI 10.6028/NIST.SP.800-56Ar2, NIST Special Publication
              800-56A, Revision 2, May 2013,
              <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/
              NIST.SP.800-56Ar2.pdf>.

Acknowledgments

Author's Address

   Jim Schaad
   August Cellars

   Email: ietf@augustcellars.com