Network Working Group   J. Schaad
Internet-Draft   August Cellars
Intended status: Informational   June 19, 2015
Expires: December 21, 2015

                     CBOR Encoded Message Syntax
                       draft-schaad-cose-msg-01

Abstract

   Concise Binary Object Representation (CBOR) is data format designed
   for small code size and small message size. There is a need for the
   ability to have the basic security services defined for this data
   format. This document specifies how to do signatures, message
   authentication codes and encryption using this data format.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on December 21, 2015.

Table of Contents

1. Introduction

There has been an increased focus on the small, constrained devices
that make up the Internet of Things (IOT). One of the standards that
has come of of this process is the Concise Binary Object
Representation (CBOR). This standard extends the data model of the
JavaScript Object Notation (JSON) by allowing for binary data among
other changes. CBOR is being adopted by several of the IETF working
groups dealing with the IOT world to do their encoding of data
structures. CBOR was designed specifically to be both small in terms
of messages transport and implementation size. A need exists to
provide basic message security services for IOT and using CBOR as the
message encoding format makes sense.

The JOSE working group produced a set of documents
[RFC7515][RFC7516][RFC7517][RFC7518] that defined how to perform
encryption, signatures and message authentication (MAC) operations
for JavaScript Object Notation (JSON) documents and then to encode
the results using the JSON format [RFC7159]. This document does the
same work for use with the Concise Binary Object Representation
(CBOR) [RFC7049] document format. While there is a strong attempt to
keep the flavor of the original JOSE documents, two considerations
are taken into account:

o CBOR has capabilities that are not present in JSON and should be
   used. One example of this is the fact that CBOR has a method of
   encoding binary directly without first converting it into a base64
   encoded string.

o The author did not always agree with some of the decisions made by
   the JOSE working group. Many of these decisions have been re-
   examined, and where it seems to the author to be superior or
   simpler, replaced.

1.1. Design changes from JOSE

o Define a top level message structure so that encrypted, signed and
   MAC-ed messages can easily identified and still have a consistent
   view.

o Signed messages separate the concept of protected and unprotected
   attributes that are for the content and the signature.

o Key management has been made to be more uniform. All key
   management techniques are represented as a recipient rather than
   only have some of them be so.

o MAC messages are separated from signed messages.

o MAC messages have the ability to do key management on the MAC
   authentication key.

o Use binary encodings for binary data rather than base64url
   encodings.

o Combine the authentication tag for encryption algorithms with the
   ciphertext.

o Remove the flattened mode of encoding. Forcing the use of an
   array of recipients at all times forces the message size to be two
   bytes larger, but one gets a corresponding decrease in the
   implementation size that should compensate for this. [CREF1]

1.2. 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
[RFC2119].

When the words appear in lower case, their natural language meaning
is used.

1.3. CBOR Grammar

There currently is no standard CBOR grammar available for use by
specifications. In this document, we use the grammar defined in the
CBOR data definition language (CDDL)
[I-D.greevenbosch-appsawg-cbor-cddl].

CDDL productions that together define the grammar are interspersed in
the document like this:

start = COSE_MSG

Note that, fromthe XML version of the present document, the collected
CDDL can be extracted via the following XPath expression:

//artwork[@type='CDDL']/text()

NOTE: At some point we need to make some decisions about how we are
using CDDL in this document. Since this draft has not been moving
forward at a great rate, changing all references on it to
informational is a good idea. On the other hand, having some type of
syntax that can be examined by a machine to do syntax checking is a
big win. The build system for this draft is currently using the
latest version of CDDL to check that the syntax of the examples is
correct. Doing this has found problems in both the syntax checker,
the syntax and the examples.

1.4. CBOR Related Terminology

In JSON, maps are called objects and only have one kind of map key: a
string. In COSE, we use both strings and integers (both positive and
negative integers) as map keys, as well as data items to identify
specific choices. The (positive and negative) integers are used for
compactness of encoding and easy comparison. Since the work "key" is
mainly used in its other meaning, as a cryptographic key, we use the
term "label" for this usage of either an integer or a string to
identify map keys and choice data items.

```
label = int / tstr
```

2. The COSE_MSG structure

The COSE_MSG structure is a top level CBOR object which corresponds
to the DataContent type in the Cryptographic Message Syntax
(CMS)[RFC5652]. This structure allows for a top level message to be
sent which could be any of the different security services. The
security service is identified within the message.

The COSE_Tagged_MSG CBOR type takes the COSE_MSG and prepends a CBOR
tag of TBD1 to the encoding of COSE_MSG. By having both a tagged and
untagged version of the COSE_MSG structure, it becomes easy to either
use COSE_MSG as a top level object or embedded in another object.
The tagged version allows for a method of placing the COSE_MSG
structure into a choice, using a consistent tag value to determine
that this is a COSE object.

The existence of the COSE_MSG and COSE_Tagged_MSG CBOR data types are
not intended to prevent protocols from using the individual security
primitives directly. Where only a single service is required, that
structure can be used directly.

Each of the top-level security objects use a CBOR map as the base
structure. Items in the map at the top level are identified by a

label. This document defines a number of labels in the IANA "COSE
Object Labels Registry" (defined in Section 9.2).

The set of labels present in a security object is not restricted to
those defined in this document. However, it is not recommended that
additional fields be added to a structure unless this is going to be
done in a closed environment. When new fields need to be added, it
is recommended that a new message type be created so that processing
of the field can be ensured. Using an older structure with a new
field means that any security properties of the new field will not be
enforced. Before a new field is added at the outer level, strong
consideration needs to be given to defining a new header field and
placing it into the protected headers. Applications should make a
determination if non-standardized fields are going to be permitted.
It is suggested that libraries allow for an option to fail parsing if
non-standardized fields exist, this is especially true if they do not
allow for access to the fields in other ways.

A field 'msg_type' is defined to distinguish between the different
structures when they appear as part of a COSE_MSG object. [CREF2]
[CREF3] This field is indexed by an integer value 1, the values
defined in this document are:

     0 - Reserved.

     1 - Signed Message.

     2 - Encrypted Message

     3 - Authenticated Message (MAC-ed message)

Implementations MUST be prepared to find an integer under this label
which does not correspond to the values 1 to 3. If this is found
then the client MUST stop attempting to parse the structure and fail.
The value of 0 is reserved and not to be used. If the value of 0 is
found, then clients MUST fail processing the structure.
Implementations need to recognize that the set of values might be
extended at a later date, but they should not provide a security
service based on guesses of what is there.

NOTE: Is the any reason to allow for a marker of a COSE_Key structure
and all it to be a COSE_MSG, doing so does allow for a security risk,
but may simplify the code. [CREF4]

The CDDL grammar that corresponds to the above is:

```
COSE_MSG = COSE_Sign /
    COSE_encrypt /
    COSE_mac

COSE_Tagged_MSG = #6.999(COSE_MSG)   ; Replace 999 with TBD1

; msg_type values
reserved=0
msg_type_signed=1
msg_type_encrypted=2
msg_type_mac=3
```

The top level of each of the COSE message structures are encoded as
maps. We use an integer to distinguish between the different
security message types. By searching for the integer under the label
identified by msg_type (which is in turn an integer), one can
determine which security message is being used and thus what syntax
is for the rest of the elements in the map.

name	number	comments
msg_type	1	Occurs only in top level messages
protected	2	Occurs in all structures
unprotected	3	Occurs in all structures
payload	4	Contains the content of the structure
signatures	5	For COSE_Sign - array of signatures
signature	6	For COSE_signature only
ciphertext	4	TODO: Should we re-use the same as payload or not?
recipients	9	For COSE_encrypt and COSE_mac
tag	10	For COSE_mac only

Table 1: COSE Map Labels

The CDDL grammar that provides the label values is:

```
; message_keys
msg_type=1
protected=2
unprotected=3
payload=4
signatures=5
signature=6
ciphertext=4
recipients=9
tag=10
```

3. Header Parameters

The structure of COSE has been designed to have two buckets of
information that are not considered to be part of the payload itself,
but are used for holding information about algorithms, keys, or
evaluation hints for the processing of the layer. These two buckets
are available for use in all of the structures in this document
except for keys. While these buckets can be present, they may not
all be usable in all instances. For example, while the protected
bucket is present for recipient structures, most of the algorithms
that are used for recipients do not provide the necessary
functionality to provide the needed protection and thus the element
is not used.

Both buckets are implemented as CBOR maps. The map key is a 'label'
Section 1.4. The value portion is dependent on the definition for
the label. Both maps use the same set of label/value pairs. The
integer range for labels has been divided into several sections with
a standard range, a private range, and a range that is dependent on
the algorithm selected. The tables of labels can be found in
Table 2.

Two buckets are provided for each layer: [CREF5]

protected contains attributes about the layer which are to be
   cryptographically protected. This bucket MUST NOT be used if it
   is not going to be included in a cryptographic computation.

unprotected contains attributes about the layer which are not
   cryptographically protected.

Both of the buckets are optional and are omitted if there are no
items contained in the map. The CDDL fragment which describes the
two buckets is:

```
header_map = {+ label => any }

Headers = (
    ? protected => bstr,
    ? unprotected => header_map
)
```

3.1. COSE Headers

The set of header fields defined in this document are:

alg This field is used to indicate the algorithm used for the
    security processing. This field MUST be present at each level of
    a signed, encrypted or authenticated message. This field using
    the integer '1' for the label. The value is taken from the 'COSE
    Algorithm Registry' (see Section 9.4).

crit This field is used to ensure that applications will take
    appropriate action based on the values found. This field uses the
    integer '2' for the label. The value is an array of COSE Header
    Labels. The field is used to indicate which protected header keys
    an application which is processing a message is required to
    understand.

        Integer keys in the range of 0 to 10 SHOULD be omitted.

        Integer keys in the range -1 to -255 can be omitted as they are
        algorithm dependent. If an application can correctly process
        an algorithm, it can be assumed that it will correctly process
        all of the parameters associated with that algorithm.

    The header values indicated by 'crit' can be processed by either
    the security library code or by an application using a security
    library, the only requirement is that the field is processed.

cty This field is used to indicate the content type of the data in
    the payload or ciphertext fields. The field uses the integer of
    '3' for the key value. The value can be either an integer or a
    string. Integers are from the XXXXX[CREF6] IANA registry table.
    Strings are from the IANA 'mime-content types' registry.
    Applications SHOULD provide this field if the content structure is
    potentially ambiguous.

kid This field one of the ways that can be used to find the key to
    be used. This value can be matched against the 'kid' field in a
    COSE_Key structure. Applications MUST NOT assume that 'kid'
    values are unique. There may be more than one key with the same
    'kid' value, it may be required that all of the keys need to be

checked to find the correct one. This field uses the integer
value of '4' for the key value. The value of field is the CBOR
'bstr' type. The internal structure of 'kid' is not defined and
generally cannot be relied on by applications.

This table contains a list of all of the parameters for use in
signature and encryption message types defined by the JOSE document
set. In the table is the data value type to be used for CBOR as well
as the integer value that can be used as a replacement for the name
in order to further decrease the size of the sent item.

name	label	value	registry	description
alg	1	int / tstr	COSE Algorithm Registry	Integers are taken from table [Appendix E](#)
crit	2	[+ label]	COSE Header Label Registry	integer values are from this table.
cty	3	tstr / int		Value is either a mime-content type or an integer from the mime-content type table
jku	*	tstr		URL to COSE key object
jwk	*	COSE_Key		contains a COSE key not a JWK key
kid	4	bstr		key identifier
x5c	*	bstr*		X.509 Certificate Chain
x5t	*	bstr		SHA-1 thumbprint of key
x5t#S256	*	bstr		SHA-256 thumbprint of key
x5u	*	tstr		URL for X.509 certificate
zip	*	int / tstr		Integers are taken from the table [Appendix E](#)

Table 2: Header Keys

OPEN ISSUES:

1. Which of the following items do we want to have standardized in
    this document: jku, jwk, x5c, x5t, x5t#S256, x5u, zip

2. I am currently torn on the question "Should epk and iv/nonce be
    algorithm specific or generic headers?" They are really specific
    to an algorithm and can potentially be defined in different ways

for different algorithms. As an example, it would make sense to
defined nonce for CCM and GCM modes that can have the leading
zero bytes stripped, while for other algorithms this might be
undesirable.

3. We might want to define some additional items. What are they? A
possible example would be a sequence number as this might be
common. On the other hand, this is the type of things that is
frequently used as the nonce in some places and thus should not
be used in the same way. Other items might be challenge/response
fields for freshness as these are likely to be common.

4. Signing Structure

The signature structure allows for one or more signatures to be
applied to a message payload. There are provisions for attributes
about the content and attributes about the signature to be carried
along with the signature itself. These attributes may be
authenticated by the signature, or just present. Examples of
attributes about the content would be the type of content, when the
content was created, and who created the content. Examples of
attributes about the signature would be the algorithm and key used to
create the signature, when the signature was created, and counter-
signatures.

When more than one signature is present, the successful validation of
one signature associated with a given signer is usually treated as a
successful signature by that signer. However, there are some
application environments where other rules are needed. An
application that employs a rule other than one valid signature for
each signer must specify those rules. Also, where simple matching of
the signer identifier is not sufficient to determine whether the
signatures were generated by the same signer, the application
specification must describe how to determine which signatures were
generated by the same signer. Support of different communities of
recipients is the primary reason that signers choose to include more
than one signature. For example, the COSE_Sign structure might
include signatures generated with the RSA signature algorithm and
with the Elliptic Curve Digital Signature Algorithm (ECDSA) signature
algorithm. This allows recipients to verify the signature associated
with one algorithm or the other. (The original source of this text
is [RFC5652].) More detailed information on multiple signature
evaluation can be found in [RFC5752].

The CDDL grammar for a signature message is:

```
COSE_Sign = {
    msg_type => msg_type_signed,
    Headers,
    ? payload => bstr,
    signatures => [+ COSE_signature]
}
```

The fields is the structure have the following semantics:

msg_type identifies this as providing the signed security service.
   The value MUST be msg_type_signed (1).

protected contains attributes about the payload which are to be
   protected by the signature. An example of such an attribute would
   be the content type ('cty') attribute. The content is a CBOR map
   of attributes which is encoded to a byte stream. This field MUST
   NOT contain attributes about the signature, even if those
   attributes are common across multiple signatures. The labels in
   this map are typically taken from Table 2.

unprotected contains attributes about the payload which are not
   protected by the signature. An example of such an attribute would
   be the content type ('cty') attribute. This field MUST NOT
   contain attributes about a signature, even if the attributes are
   common across multiple signatures. The labels in this map are
   typically taken from Table 2.

payload contains the serialized content to be signed.
   If the payload is not present in the message, the application is
   required to supply the payload separately.
   The payload is wrapped in a bstr to ensure that it is transported
   without changes, if the payload is transported separately it is
   the responsibility of the application to ensure that it will be
   transported without changes.

signatures is an array of signature items. Each of these items uses
   the COSE_signature structure for its representation.

We use the values in Table 1 as the labels in the COSE_Sign map.
While other labels can be present in the map, it is not generally a
recommended practice. The other labels can be either of integer or
string type, use of other types SHOULD be treated as an error.

The CDDL grammar structure for a signature is:

```
COSE_signature =  {
    Headers,
    signature => bstr
}
```

The fields in the structure have the following semantics:

protected contains additional information to be authenticated by the
   signature. The field holds data about the signature operation.
   The field MUST NOT hold attributes about the payload being signed.
   The content is a CBOR map of attributes which is encoded to a byte
   stream. At least one of protected and unprotected MUST be
   present.

unprotected contains attributes about the signature which are not
   protected by the signature. This field MUST NOT contain
   attributes about the payload being signed. At least one of
   protected and unprotected MUST be present.

signature contains the computed signature value.

The COSE structure used to create the byte stream to be signed uses
the following CDDL grammar structure:

```
Sig_structure = [
    body_protected: bstr,
    sign_protected: bstr,
    payload: bstr
]
```

How to compute a signature:

1. Create a Sig_structure object and populate it with the
    appropriate fields. For body_protected and sign_protected, if
    the fields are not present in their corresponding maps, an bstr
    of length zero is used.

2. Create the value ToBeSigned by encoding the Sig_structure to a
    byte string.

3. Call the signature creation algorithm passing in K (the key to
    sign with), alg (the algorithm to sign with) and ToBeSigned (the
    value to sign).

4. Place the resulting signature value in the 'signature' field of
    the map.

How to verify a signature:

1. Create a Sig_structure object and populate it with the
    appropriate fields. For body_protected and sign_protected, if
    the fields are not present in their corresponding maps, an bstr
    of length zero is used.

2. Create the value ToBeSigned by encoding the Sig_structure to a
    byte string.

3. Call the signature verification algorithm passing in K (the key
    to verify with), alg (the algorithm to sign with), ToBeSigned
    (the value to sign), and sig (the signature to be verified).

In addition to performing the signature verification, one must also
perform the appropriate checks to ensure that the key is correctly
paired with the signing identity and that the appropriated
authorization is done.

5. Encryption object

In this section we describe the structure and methods to be used when
doing an encryption in COSE. In COSE, we use the same techniques and
structures for encrypting both the plain text and the keys used to
protect the text. This is different from the approach used by both
[RFC5652] and [RFC7516] where different structures are used for the
plain text and for the different key management techniques.

One of the byproducts of using the same technique for encrypting and
encoding both the content and the keys using the various key
management techniques, is a requirement that all of the key
management techniques use an Authenticated Encryption (AE) algorithm.
(For the purpose of this document we use a slightly loose definition
of AE algorithms.) When encrypting the plain text, it is normal to
use an Authenticated Encryption with Additional Data (AEAD)
algorithm. For key management, either AE or AEAD algorithms can be
used. See Appendix A for more details about the different types of
algorithms. [CREF7]

The CDDL grammar structure for encryption is:

```
COSE_encrypt = {
    msg_type=>msg_type_encrypted,
    COSE_encrypt_fields
}

COSE_encrypt_fields = (
    Headers,
    ? ciphertext => bstr,
    ? recipients => [+{COSE_encrypt_fields}]
)
```

Description of the fields:

msg_type   identifies this as providing the encrypted security
   service. The value MUST be msg_type_encrypted (2).

protected   contains the information about the plain text or
   encryption process that is to be integrity protected. The field
   is encoded in CBOR as a 'bstr'. The contents of the protected
   field is a CBOR map of the protected data names and values. The
   map is CBOR encoded before placing it into the bstr. Only values
   associated with the current cipher text are to be placed in this
   location even if the value would apply to multiple recipient
   structures.

unprotected   contains information about the plain text that is not
   integrity protected. Only values associated with the current
   cipher text are to be placed in this location even if the value
   would apply to multiple recipient structures.

cipherText   contains the encrypted plain text. If the cipherText is
   to be transported independently of the control information about
   the encryption process (i.e. detached content) then the field is
   omitted.

recipients   contains the recipient information. It is required that
   at least one recipient MUST be present for the content encryption
   layer.

5.1. Key Management Methods

There are a number of different key management methods that can be
used in the COSE encryption system. In this section we will discuss
each of the key management methods and what fields need to be
specified to deal with each of them.

The names of the key management methods used here are the same as are
defined in [RFC7517]. Other specifications use different terms for

the key management methods or do not support some of the key
management methods.

At the moment we do not have any key management methods that allow
for the use of protected headers. This may be changed in the future
if, for example, the AES-GCM Key wrap method defined in [RFC7518]
were extended to allow for authenticated data. In that event the use
of the 'protected' field, which is current forbidden below, would be
permitted.

5.1.1. Direct Encryption

In direct encryption mode, a shared secret between the sender and the
recipient is used as the CEK. When direct encryption mode is used,
it MUST be the only mode used on the message. It is a massive
security leak to have both direct encryption and a different key
management mode on the same message.

For JOSE, direct encryption key management is the only key management
method allowed for doing MAC-ed messages. In COSE, all of the key
management methods can be used for MAC-ed messages.

The COSE_encrypt structure for the recipient is organized as follows:

o The 'protected', 'ciphertext' and 'recipients' fields MUST be
   absent.

o At a minimum, the 'unprotected' field MUST contain the 'alg'
   parameter and SHOULD contain a parameter identifying the shared
   secret.

5.1.2. Key Wrapping

In key wrapping mode, the CEK is randomly generated and that key is
then encrypted by a shared secret between the sender and the
recipient. All of the currently defined key wrapping algorithms for
JOSE (and thus for COSE) are AE algorithms. Key wrapping mode is
considered to be superior to direct encryption if the system has any
capability for doing random key generation. This is because the
shared key is used to wrap random data rather than data has some
degree of organization and may in fact be repeating the same content.

The COSE_encrypt structure for the recipient is organized as follows:

o The 'protected', and 'recipients' fields MUST be absent.

o The plain text to be encrypted is the key from next layer down
   (usually the content layer).

o At a minimum, the 'unprotected' field MUST contain the 'alg'
   parameter and SHOULD contain a parameter identifying the shared
   secret.

5.1.3. Key Encryption

Key Encryption mode is also called key transport mode in some
standards. Key Encryption mode differs from Key Wrap mode in that it
uses an asymmetric encryption algorithm rather than a symmetric
encryption algorithm to protect the key. The only current Key
Encryption mode algorithm supported is RSAES-OAEP.

The COSE_encrypt structure for the recipient is organized as follows:

o The 'protected' field MUST be absent.

o The plain text to be encrypted is the key from next layer down
   (usually the content layer).

o At a minimum, the 'unprotected' field MUST contain the 'alg'
   parameter and SHOULD contain a parameter identifying the
   asymmetric key.

5.1.4. Direct Key Agreement

Direct Key Agreement derives the CEK from the shared secret computed
by the key agreement operation.

When direct key agreement mode is used, it SHOULD be the only mode
used on the message. This method creates the CEK directly and that
makes it difficult to mix with additional recipients.

The COSE_encrypt structure for the recipient is organized as follows:

o The 'protected' field MUST be absent.

o At a minimum, the 'unprotected' field MUST contain the 'alg'
   parameter and SHOULD contain a parameter identifying the
   asymmetric key.

o The 'unprotected' field MUST contain the 'epk' parameter.

5.1.5. Key Agreement with Key Wrapping

Key Agreement with Key Wrapping uses a randomly generated CEK. The
CEK is then encrypted using a Key Wrapping algorithm and a key
derived from the shared secret computed by the key agreement
algorithm.

The COSE_encrypt structure for the recipient is organized as follows:

o The 'protected' field MUST be absent.

o The plain text to be encrypted is the key from next layer down
   (usually the content layer).

o At a minimum, the 'unprotected' field MUST contain the 'alg'
   parameter, a parameter identifying the recipient asymmetric key,
   and a parameter with the sender's asymmetric public key.

5.2. Encryption Algorithm for AEAD algorithms

The encryption algorithm for AEAD algorithms is fairly simple.
In order to get a consistent encoding of the data to be
authenticated, the Enc_structure is used to have canonical form of
the AAD.

```
Enc_structure = [
    protected: bstr,
    external_aad: bstr
]
```

1. Copy the protected header field from the message to be sent.

2. If the application has supplied external additional authenticated
    data to be included in the computation, then it is placed in the
    'external_aad' field. If no data was supplied, then a zero
    length binary value is used.

3. Encode the Enc_structure using a CBOR Canonical encoding
    Section 8 to get the AAD value.

4. Determine the encryption key. This step is dependent on the key
    management method being used: For:

    No Recipients: The key to be used is determined by the algorithm
       and key at the current level.

    Direct and Direct Key Agreement: The key is determined by the
       key and algorithm in the recipient structure. The encryption
       algorithm and size of the key to be used are inputs into the
       KDF used for the recipient. (For direct, the KDF can be
       thought of as the identity operation.)

    Other: The key is randomly generated.

5. Call the encryption algorithm with K (the encryption key to use),
    P (the plain text) and AAD (the additional authenticated data).
    Place the returned cipher text into the 'ciphertext' field of the
    structure.

6. For recipients of the message, recursively perform the encryption
    algorithm for that recipient using the encryption key as the
    plain text.

5.3. Encryption algorithm for AE algorithms

1. Verify that the 'protected' field is absent.

2. Verify that there was no external additional authenticated data
    supplied for this operation.

3. Determine the encryption key. This step is dependent on the key
    management method being used: For:

    No Recipients: The key to be used is determined by the algorithm
       and key at the current level.

    Direct and Direct Key Agreement: The key is determined by the
       key and algorithm in the recipient structure. The encryption
       algorithm and size of the key to be used are inputs into the
       KDF used for the recipient. (For direct, the KDF can be
       thought of as the identity operation.)

    Other: The key is randomly generated.

4. Call the encryption algorithm with K (the encryption key to use)
    and the P (the plain text). Place the returned cipher text into
    the 'ciphertext' field of the structure.

5. For recipients of the message, recursively perform the encryption
    algorithm for that recipient using the encryption key as the
    plain text.

6. MAC objects

In this section we describe the structure and methods to be used when
doing MAC authentication in COSE. JOSE used a variant of the
signature structure for doing MAC operations and it is restricted to
using a single pre-shared secret to do the authentication. This
document allows for the use of all of the same methods of key
management as are allowed for encryption.

When using MAC operations, there are two modes in which it can be
used. The first is just a check that the content has not been
changed since the MAC was computed. Any of the key management
methods can be used for this purpose. The second mode is to both
check that the content has not been changed since the MAC was
computed, and to use key management to verify who sent it. The key
management modes that support this are ones that either use a pre-
shared secret, or do static-static key agreement. In both of these
cases the entity MAC-ing the message can be validated by a key
binding. (The binding of identity assumes that there are only two
parties involved and you did not send the message yourself.)

```
COSE_mac = {
   msg_type=>msg_type_mac,
   Headers,
   ? payload => bstr,
   tag => bstr,
   recipients => [+{COSE_encrypt_fields}]
}
```

Field descriptions:

msg_type identifies this as providing the encrypted security
   service. The value MUST be msg_type_mac (3).

protected contains attributes about the payload which are to be
   protected by the MAC. An example of such an attribute would be
   the content type ('cty') attribute. The content is a CBOR map of
   attributes which is encoded to a byte stream. This field MUST NOT
   contain attributes about the recipient, even if those attributes
   are common across multiple recipients. At least one of protected
   and unprotected MUST be present.

unprotected contains attributes about the payload which are not
   protected by the MAC. An example of such an attribute would be
   the content type ('cty') attribute. This field MUST NOT contain
   attributes about a recipient, even if the attributes are common
   across multiple recipients. At least one of protected and
   unprotected MUST be present.

payload contains the serialized content to be MAC-ed.
   If the payload is not present in the message, the application is
   required to supply the payload separately.
   The payload is wrapped in a bstr to ensure that it is transported
   without changes, if the payload is transported separately it is
   the responsibility of the application to ensure that it will be
   transported without changes.

   tag contains the MAC value.

   recipients contains the recipient information. See the description
      under COSE_Encryption for more info.

```
 MAC_structure = [
     protected: bstr,
     external_aad: bstr,
     payload: bstr
]
```

   How to compute a MAC:

   1. Create a MAC_structure and copy the protected and payload
       elements from the COSE_mac structure.

   2. If the application has supplied external authenticated data,
       encode it as a binary value and place in the MAC_structure. If
       there is no external authenticated data, then use a zero length
       'bstr'.

   3. Encode the MAC_structure using a canonical CBOR encoder. The
       resulting bytes is the value to compute the MAC on.

   4. Compute the MAC and place the result in the 'tag' field of the
       COSE_mac structure.

   5. Encrypt and encode the MAC key for each recipient of the message.

7. Key Structure

   There are only a few changes between JOSE and COSE for how keys are
   formatted. As with JOSE, COSE uses a map to contain the elements of
   a key. Those values, which in JOSE, are base64url encoded because
   they are binary values, are encoded as bstr values in COSE.

   For COSE we use the same set of fields that were defined in
   [RFC7517].

```
COSE_Key = {
    kty => tstr / int,
    ? key_ops => [+ tstr / int ],
    ? alg => tstr / int,
    ? kid => bstr,
    * keys => values
}

COSE_KeySet = [+COSE_Key]
```

The element "kty" is a required element in a COSE_Key map.
All other elements are optional and not all of the elements listed in
[RFC7517] or [RFC7518] have been listed here even though they can all
appear in a COSE_Key map.

The "key_ops" element is preferred over the "use" element as the
information provided that way is more finely detailed about the
operations allowed. It is strongly suggested that this element be
present for all keys.

The same fields defined in [RFC7517] are used here with the following
changes in rules:

o Any item which is base64 encoded in JWK, is bstr encoded for COSE.

o Any item which is integer encoded in JWK, is int encoded for COSE.

o Any item which is string (but not base64) encoded in JWK, is tstr
   encoded for COSE.

o Exceptions to this are the following fields:

   kid is always bstr encoded rather than tstr encoded. This change
      in encoded is due to the fact that frequently, values such as a
      hash of the public key is used for a kid value. Since the
      field is defined as not having a specific structure, making it
      binary rather than textual makes sense.

8. CBOR Encoder Restrictions

   There as 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:

   o The restriction applies to the encoding the Sig_structure, the
      Enc_structure, and the MAC_structure.

o The rules for Canonical CBOR (Section 3.9 of RFC 7049) MUST be
   used in these locations. The main rule that needs to be enforced
   is that all lengths in these structures MUST be encoded such that
   they are encoded using definite lengths and the minimum length
   encoding is used.

o All parsers used SHOULD fail on both parsing and generation if the
   same label is used twice as a key for the same map.

9. IANA Considerations

9.1. CBOR Tag assignment

It is requested that IANA assign a new tag from the "Concise Binary
Object Representation (CBOR) Tags" registry. It is requested that
the tag be assigned in the 0 to 23 value range.

Tag Value: TBD1

Data Item: CBOR map

Semantics: COSE security message.

9.2. COSE Object Labels Registry

It is requested that IANA create a new registry entitled "COSE Object
Labels Registry". [CREF8]

This table is initially populated by the table in Table 1.

9.3. COSE Header Label Table

It is requested that IANA create a new registry entitled "COSE Header
Labels".

The columns of the registry are:

name The name is present to make it easier to refer to and discuss
   the registration entry. The value is not used in the protocol.
   Names are to be unique in the table.

label This is the value used for the label. The label can be either
   an integer or a string. Registration in the table is based on the
   value of the label requested. Integer values between 0 and 255
   and strings of length 1 are designated as Standards Track Document
   required. Integer values from 256 to 65535 and strings of length
   2 are designated as Specification Required. Integer values of
   greater than 65535 and strings of length greater than 2 are

designated as first come first server. Integer values in the
range -1 to -65536 are delegated to the "COSE Header Algorithm
Label" registry. Integer values beyond -65536 are marked as
private use.

value This contains the CBOR type for the value portion of the
   label.

value registry This contains a pointer to the registry used to
   contain values where the set is limited.

description This contains a brief description of the header field.

specification This contains a pointer to the specification defining
   the header field (where public).

The initial contents of the registry can be found in Table 2. The
specification column for all rows in that table should be this
document.

NOTE: Need to review the range assignments. It does not necessarily
make sense as specification required uses 1 byte positive integers
and 2 byte strings.

9.4. COSE Header Algorithm Label Table

It is requested that IANA create a new registry entitled "COSE Header
Algorithm Labels".

The columns of the registry are:

name The name is present to make it easier to refer to and discuss
   the registration entry. The value is not used in the protocol.

algorithm The algorithm(s) that this registry entry is used for.
   This value is taken from the "COSE Algorithm Value" registry.
   Multiple algorithms can be specified in this entry. For the
   table, the algorithm, label pair MUST be unique.

label This is the value used for the label. The label is an integer
   in the range of -1 to -65536.

value This contains the CBOR type for the value portion of the
   label.

value registry This contains a pointer to the registry used to
   contain values where the set is limited.

   description This contains a brief description of the header field.

   specification This contains a pointer to the specification defining
      the header field (where public).

   The initial contents of the registry can be found in Appendix D. The
   specification column for all rows in that table should be this
   document.

9.5. COSE Algorithm Registry

   It is requested that IANA create a new registry entitled "COSE
   Algorithm Registry".

   The columns of the registry are:

   value The value to be used to identify this algorithm. Algorithm
      values MUST be unique. The value can be a positive integer, a
      negative integer or a string. Integer values between 0 and 255
      and strings of length 1 are designated as Standards Track Document
      required. Integer values from 256 to 65535 and strings of length
      2 are designated as Specification Required. Integer values of
      greater than 65535 and strings of length greater than 2 are
      designated as first come first server. Integer values in the
      range -1 to -65536 are delegated to the "COSE Header Algorithm
      Label" registry. Integer values beyond -65536 are marked as
      private use.

   description A short description of the algorithm.

   specification A document where the algorithm is defined (if publicly
      available).

   The initial contents of the registry can be found in Appendix E. The
   specification column for all rows in that table should be this
   document.

9.6. COSE Key Map Registry

   It is requested that IANA create a new registry entitled "COSE Key
   Map Registry".

   The columns of the registry are:

   name This is a descriptive name that enables easier reference to the
      item. It is not used in the encoding.

label The value to be used to identify this algorithm. Algorithm
   labels MUST be unique. The label can be a positive integer, a
   negative integer or a string. Integer values between 0 and 255
   and strings of length 1 are designated as Standards Track Document
   required. Integer values from 256 to 65535 and strings of length
   2 are designated as Specification Required. Integer values of
   greater than 65535 and strings of length greater than 2 are
   designated as first come first server. Integer values in the
   range -1 to -65536 are used for key parameters specific to a
   single algorithm delegated to the "COSE Key Parameter Label"
   registry. Integer values beyond -65536 are marked as private use.

CBOR Type This field contains the CBOR type for the field

registry This field denotes the registry that values come from, if
   one exists.

description This field contains a brief description for the field

specification This contains a pointer to the public specification
   for the field if one exists

This registry will be initially populated by the values in
Appendix G. The specification column for all of these entries will
be this document.

9.7. COSE Key Parameter Registry

It is requested that IANA create a new registry "COSE Key
Parameters".

The columns of the table are:

key type This field contains a descriptive string of a key type.
   This should be a value that is in the COSE General Values table
   and is placed in the 'kty' field of a COSE Key structure.

name This is a descriptive name that enables easier reference to the
   item. It is not used in the encoding.

label The label is to be unique for every value of key type. The
   range of values is from -256 to -1. Keys are expected to be re-
   used for different keys.

CBOR type This field contains the CBOR type for the field

description This field contains a brief description for the field

specification This contains a pointer to the public specification
   for the field if one exists

This registry will be initially populated by the values in
[Appendix H](#). The specification column for all of these entries will
be this document.

9.8. Media Type Registration

9.8.1. COSE Security Message

This section registers the "application/cose" and "application/
cose+cbor" media types in the "Media Types" registry. [[CREF9](#)] These
media types are used to indicate that the content is a COSE_MSG.

   Type name: application

   Subtype name: cose

   Required parameters: N/A

   Optional parameters: N/A

   Encoding considerations: binary

   Security considerations: See the Security Considerations section
   of RFC TBD.

   Interoperability considerations: N/A

   Published specification: RFC TBD

   Applications that use this media type: To be identified

   Fragment identifier considerations: N/A

   Additional information:

   * Magic number(s): N/A

   * File extension(s): cbor

   * Macintosh file type code(s): N/A

   Person & email address to contact for further information:
   iesg@ietf.org

   Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

Type name: application

Subtype name: cose+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section
of RFC TBD.

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

* Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

9.8.2. COSE Key media type

This section registers the "application/jwk+json" and "application/
jwk-set+json" media typesin the "Media Types" registry. These media
types are used to indicate, respectively, that content is a COSE_Key
or COSE_KeySet object.

Type name: application

Subtype name: cose-key+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section
of RFC TBD.

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

* Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

Type name: application

Subtype name: cose-key-set+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section
of RFC TBD.

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

* Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

10. Security Considerations

There are security considerations:

1. Protect private keys

2. MAC messages with more than one recipient means one cannot figure
    out who sent the message

3. Use of direct key with other recipient structures hands the key
    to other recipients.

4. Use of direct ECDH direct encryption is easy for people to leak
    information on if there are other recipients in the message.

5. Considerations about protected vs unprotected header fields.

11. References

11.1. Normative References

[I-D.greevenbosch-appsawg-cbor-cddl]
          Vigano, C., Birkholz, H., and R. Sun, "CBOR data
          definition language: a notational convention to express
          CBOR data structures.", draft-greevenbosch-appsawg-cbor-
          cddl-05 (work in progress), March 2015.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
          Representation (CBOR)", RFC 7049, October 2013.

11.2. Informative References

[AES-GCM] Dworkin, M., "NIST Special Publication 800-38D:
          Recommendation for Block Cipher Modes of Operation:
          Galois/Counter Mode (GCM) and GMAC.", June 2015.

[I-D.mcgrew-aead-aes-cbc-hmac-sha2]
          McGrew, D., Foley, J., and K. Paterson, "Authenticated
          Encryption with AES-CBC and HMAC-SHA", draft-mcgrew-aead-
          aes-cbc-hmac-sha2-05 (work in progress), July 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
          Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard
          (AES) Key Wrap Algorithm", RFC 3394, September 2002.

   [RFC3447]   Jonsson, J. and B. Kaliski, "Public-Key Cryptography
               Standards (PKCS) #1: RSA Cryptography Specifications
               Version 2.1", RFC 3447, February 2003.

   [RFC3610]   Whiting, D., Housley, R., and N. Ferguson, "Counter with
               CBC-MAC (CCM)", RFC 3610, September 2003.

   [RFC5652]   Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
               RFC 5652, September 2009.

   [RFC5752]   Turner, S. and J. Schaad, "Multiple Signatures in
               Cryptographic Message Syntax (CMS)", RFC 5752, January
               2010.

   [RFC5990]   Randall, J., Kaliski, B., Brainard, J., and S. Turner,
               "Use of the RSA-KEM Key Transport Algorithm in the
               Cryptographic Message Syntax (CMS)", RFC 5990, September
               2010.

   [RFC7159]   Bray, T., "The JavaScript Object Notation (JSON) Data
               Interchange Format", RFC 7159, March 2014.

   [RFC7515]   Jones, M., Bradley, J., and N. Sakimura, "JSON Web
               Signature (JWS)", RFC 7515, May 2015.

   [RFC7516]   Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",
               RFC 7516, May 2015.

   [RFC7517]   Jones, M., "JSON Web Key (JWK)", RFC 7517, May 2015.

   [RFC7518]   Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, May
               2015.

Appendix A. AEAD and AE algorithms

   The set of encryption algorithms that can be used with this
   specification is restricted to authenticated encryption (AE) and
   authenticated encryption with additional data (AEAD) algorithms.
   This means that there is a strong check that the data decrypted by
   the recipient is the same as what was encrypted by the sender.
   Encryption modes such as counter have no check on this at all. The
   CBC encryption mode had a weak check that the data is correct, given
   a random key and random data, the CBC padding check will pass one out
   of 256 times. There have been several times that a normal encryption
   mode has been combined with an integrity check to provide a content
   encryption mode that does provide the necessary authentication. AES-
   GCM [AES-GCM], AES-CCM [RFC3610], AES-CBC-HMAC

[I-D.mcgrew-aead-aes-cbc-hmac-sha2] are examples of these composite modes.

2PKCS v1.5 RSA key transport does not qualify as an AE algorithm. There are only three bytes in the encoding that can be checked as having decrypted correctly, the rest of the content can only be probabilistically checked as having decrypted correctly. For this reason, PKCS v1.5 RSA key transport MUST NOT be used with this specification. RSA-OAEP was designed to have the necessary checks that that content correctly decrypted and does qualify as an AE algorithm.

When dealing with authenticated encryption algorithms, there is always some type of value that needs to be checked to see if the authentication level has passed. This authentication value may be:

o A separately generated tag computed by both the encrypter and decrypter and then compared by the decryptor. This tag value may be either placed at the end of the cipher text (the decision we made) or kept separately (the decision made by the JOSE working group). This is the approach followed by AES-GCM [AES-GCM] and AES-CCM [RFC3610].

o A fixed value which is part of the encoded plain text. This is the approach followed by the AES key wrap algorithm [RFC3394].

o A computed value is included as part of the encoded plain text. The computed value is then checked by the decryptor using the same computation path. This is the approach followed by RSAES-OAEP [RFC3447].

Appendix B. Three Levels of Recipient Information

All of the currently defined Key Management methods only use two levels of the COSE_Encrypt structure. The first level is the message content and the second level is the content key encryption. However, if one uses a key management technique such as RSA-KEM (see Appendix A of RSA-KEM [RFC5990], then it make sense to have three levels of the COSE_Encrypt structure.

These levels would be:

o Level 0: The content encryption level. This level contains the payload of the message.

o Level 1: The encryption of the CEK by a KEK.

o Level 2: The encryption of a long random secret using an RSA key
   and a key derivation function to convert that secret into the KEK.

This is an example of what a triple layer message would look like.
The message has the following layers:

o Level 0: Has a content encrypted with AES-GCM using a 128-bit key.

o Level 1: Uses the AES Key wrap algorithm with a 128-bit key.

o Level 3: Uses ECDH Ephemeral-Static direct to generate the level 1
   key.

In effect this example is a decomposed version of using the ECDH-
ES+A128KW algorithm.

```
   {
     1: 2,
     2: h'a10101',
     3: {
       -1: h'02d1f7e6f26c43d4868d87ce'
     },
     4: h'64f84d913ba60a76070a9a48f26e97e863e285295a44320878caceb076
   3a334806857c67',
     9: [
       {
         3: {
           1: -3
         },
         4: h'5a15dbf5b282ecb31a6074ee3815c252405dd7583e078188',
         9: [
           {
             3: {
               1: "ECDH-ES",
               5: "meriadoc.brandybuck@buckland.example",
               4: {
                 1: 1,
                 -1: 4,
                 -2: h'b2add44368ea6d641f9ca9af308b4079aeb519f11e9b8
   a55a600b21233e86e68',
                 -3: h'1a2cf118b9ee6895c8f415b686d4ca1cef362d4a7630a
   31ef6019c0c56d33de0'
               }
             }
           }
         ]
       }
     ]
   }
```

Appendix C. Examples

   The examples can be found at https://github.com/cose-wg/Examples. I
   am currently still in the process of getting the examples up there
   along with some control information for people to be able to check
   and reproduce the examples.

C.1. Direct MAC

   This example has some features that are in questions but not yet
   incorporated in the document.

   To make it easier to read, this uses CBOR's diagnostic notation
   rather than a binary dump.

This example is uses HMAC with SHA-256 as the digest algorithm. The
key management is uses two static ECDH keys along with HKDF to
directly derive the key used in the HMAC operation.

```
{
  1: 3,
  2: h'a10104',
  4: h'546869732069732074686520636f6e74656e742e',
  10: h'2ba937ca03d76c3dbad30cfcbaeef586f9c0f9ba616ad67e9205d3857
6ad9930',
  9: [
    {
      3: {
        1: "ECDH-SS",
        5: "meriadoc.brandybuck@buckland.example",
        "spk": {
          "kid": "peregrin.took@tuckborough.example"
        },
        "apu": h'4d8553e7e74f3c6a3a9dd3ef286a8195cbf8a23d19558ccf
ec7d34b824f42d92bd06bd2c7f0271f0214e141fb779ae2856abf585a58368b01
7e7f2a9e5ce4db5'
      }
    }
  ]
}
```

C.2. Wrapped MAC

This example has some features that are in questions but not yet
incorporated in the document.

To make it easier to read, this uses CBOR's diagnostic notation
rather than a binary dump.

This example uses AES-128-MAC truncated to 64-bits as the digest
algorithm. It uses AES-256 Key wrap for the key management algorithm
wrapping the 128-bit key used for the digest algorithm.

```
   {
     1: 3,
     2: h'a1016e4145532d3132382d4d41432d3634',
     4: h'54686973206973207468652074686520636f6e74656e742e',
     10: h'6d1fa77b2dd9146a',
     9: [
       {
         3: {
           1: -5,
           5: "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
         },
         4: h'711ab0dc2fc4585dce27effa6781c8093eba906f227b6eb0'
       }
     ]
   }
```

C.3. Multi-recipient MAC message

   This example has some features that are in questions but not yet
   incorporated in the document.

   To make it easier to read, this uses CBOR's diagnostic notation
   rather than a binary dump.

   This example uses HMAC with SHA-256 for the digest algorithm. There
   are three different key management techniques applied:

   o An ephemeral static ECDH key agreement operation using AES-128 key
      wrap on the digest key.

   o Key transport using RSA-OAEP with SHA-256 for the hash and the mfg
      function operations.

   o AES 256-bit Key wrap using a pre-shared secret.

```
   {
     1: 3,
     2: h'a10104',
     4: h'54686973206973207468652074686520636f6e74656e742e',
     10: h'7aaa6e74546873061f0a7de21ff0c0658d401a68da738dd8937486519
   83ce1d0',
     9: [
       {
         3: {
           1: "ECDH-ES+A128KW",
           5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
   6d706c65',
           4: {
```

```
            1: 1,
            -1: 5,
            -2: h'43b12669acac3fd27898ffba0bcd2e6c366d53bc4db71f909
   a759304acfb5e18cdc7ba0b13ff8c7636271a6924b1ac63c02688075b55ef2d61
   3574e7dc242f79c3',
            -3: h'812dd694f4ef32b11014d74010a954689c6b6e8785b333d1a
   b44f22b9d1091ae8fc8ae40b687e5cfbe7ee6f8b47918a07bb04e9f5b1a51a334
   a16bc09777434113'
          }
        },
        4: h'1b120c848c7f2f8943e402cbdbdb58efb281753af4169c70d0126c
   0d16436277160821790ef4fe3f'
      },
      {
        3: {
          1: -2,
          5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
   6d706c65'
        },
        4: h'46c4f88069b650909a891e84013614cd58a3668f88fa18f3852940
   a20b35098591d3aacf91c125a2595cda7bee75a490579f0e2f20fd6bc956623bf
   de3029c318f82c426dac3463b261c981ab18b72fe9409412e5c7f2d8f2b5abaf7
   80df6a282db033b3a863fa957408b81741878f466dcc437006ca21407181a016c
   a608ca8208bd3c5a1ddc828531e30b89a67ec6bb97b0c3c3c92036c0cb84aa0f0
   ce8c3e4a215d173bfa668f116ca9f1177505afb7629a9b0b5e096e81d37900e06
   f561a32b6bc993fc6d0cb5d4bb81b74e6ffb0958dac7227c2eb8856303d989f93
   b4a051830706a4c44e8314ec846022eab727e16ada628f12ee7978855550249cc
   b58'
      },
      {
        3: {
          1: -5,
          5: "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
        },
        4: h'0b2c7cfce04e98276342d6476a7723c090dfdd15f9a518e7736549
   e998370695e6d6a83b4ae507bb'
      }
    ]
  }
```

C.4. Direct ECDH

This example has some features that are in questions but not yet
incorporated in the document.

To make it easier to read, this uses CBOR's diagnostic notation
rather than a binary dump.

   Encoded in CBOR - 216 bytes, content is 14 bytes long

   {
     1: 2,
     2: h'a10101',
     3: {
       -1: h'c9cf4df2fe6c632bf7886413'
     },
     4: h'45fce2814311024d3a479e7d3eed063850f3f0b9f3f948677e3ae9869b
   cf9ff4e1763812',
     9: [
       {
          3: {
           1: "ECDH-ES",
           5: "meriadoc.brandybuck@buckland.example",
           4: {
             1: 1,
             -1: 4,
             -2: h'98f50a4ff6c05861c8860d13a638ea56c3f5ad7590bbfbf05
   4e1c7b4d91d6280',
             -3: h'f01400b089867804b8e9fc96c3932161f1934f4223069170d
   924b7e03bf822bb'
           }
         }
       }
     ]
   }

C.5. Single Signature

   This example has some features that are in questions but not yet
   cooperated in the document.

   To make it easier to read, this uses CBOR's diagnostic notation
   rather than a binary dump.

```
  {
    1: 1,
    4: h'54686973206973207468652063f6e74656e742e',
    5: [
      {
         2: h'a20165505333383405781e62696c626f2e62616767696e7340686f
  626269746f6e2e6578616d706c65',
         6: h'7c4656acc11ffe98e2ea4babff6d177b2e5a088da2034e0096a6f8
  8cd50a36ed971a83f42244d40d97043d080f43dff7a7c0eb5bc322a3d3d18826c
  e755d82293ecc22a3919857bc60c456017ca87b7d662971687aac7315f68c9f52
  d95c24f72418b7eb07cd432875ab42658bde269534da10d8572c2d5340be660cf
  cf7f6cceaf5cd02ddfdac1cf18930199c0e8721c2d71c646516cc3da79f6d555e
  89825803ef1ecf7fb411cea24d15610f56d33af0d1a7e68a2e07e9790326f01f2
  9ad7210e317c637f7cbd16f8cd32a9959dfd419de697dfa3145aa7cd1019072f9
  5c3d041bd8d47de09abbce16117733378e1593fe2439454cd907f88aabc664d26
  298'
      }
    ]
  }
```

C.6. Multiple Signers

This example has some features that are in questions but not yet
cooperated in the document.

To make it easier to read, this uses CBOR's diagnostic notation
rather than a binary dump.

Encoded in CBOR - 491 bytes, content is 14 bytes long

```
   {
     1: 1,
     4: h'546869732069732074686520636f6e74656e742e',
     5: [
       {
         2: h'a10129',
         3: {
           5: "bilbo.baggins@hobbiton.example"
         },
         6: h'93f83aaf872d9fc8c7cad30437dc8fed33a673ec5d4d8e004f16d9
   ed236f6b2ce4cc15ea48aef8c5c0c4eab8ed539a4eae27ac9d5f5e6ef7b9cfb02
   133f3f4ad7062db989fcff6ecc67c13624418224416116ff0a67d1a133d27bb79
   1a1893a03c683d84def6742059c63670c2738efc2dde8eac364b4d714c60db424
   ffbfb098c579c6538daa2a9fc5be2d829a82c148c9913a537f98ceb469b78fae2
   02531353fe740ba5ef3eb01377b01e7b27f0ffad35cdd3ef8d2c010a60331b2ac
   2924183022425a1a3a4495adb03d67a92fb420bd408cb0c81d5bac694d1568f11
   2ec8bbab834ca02b098c3b8f4975d75114bb001408252875d14bc1516f843b9fa
   bb0'
       },
       {
         3: {
           1: -9,
           5: "bilbo.baggins@hobbiton.example"
         },
         6: h'5bacf5cf8e84051f01030a688c9c5dfa867173a6038e6655374ce3
   e07de671d16dbe8cee3e965f2492d7850debf63b64a93e8fe7062ea536f3f165a
   e34d5852f5001f1a1caffc52b05aec184da5a4148e305d87d0ef68642701b05c6
   dde19669a1ffb39158a8fa4d51e95a2557b86da7a64a719a1422568dec9ec699d
   af93a58eab268'
       }
     ]
   }
```

Appendix D. COSE Header Algorithm Label Table

name	algorithm	label	CBOR type	description
apu	ECDH	-1	bstr	
apv	ECDH	-2	bstr	
epk	ECDH	-3	COSE_Key	contains a COSE key not a JWK key
iv	A128GCMKW, A192GCMKW, A256GCMKW	-1	bstr	
iv	A128GCM, A192GCM, A256GCM	-1	bstr	
p2c	PBE	-1	int	
p2s	PBE	-2	bstr	

[Appendix E](#). COSE Algorithm Name Values

This table contains all of the defined algorithms for COSE.

name	value	description
HS256	4	HMAC w/ SHA-256
HS384	5	HMAC w/ SHA-384
HS512	6	HMAC w/ SHA-512
RS256	*	RSASSA-v1.5 w/ SHA-256
RS384	*	RSASSA-v1.5 w/ SHA-384
RSA512	*	RSASSA-v1.5 w/ SHA-256
ES256	-7	ECDSA w/ SHA-256
ES384	-8	ECDSA w/ SHA-384
ES512	-9	ECDSA w/ SHA-512

PS256	-10	RSASSA-PSS w/ SHA-256
PS384	*	RSASSA-PSS w/ SHA-384
PS512	-11	RSASSA-PSS w/ SHA-512
RSA1_5	*	RSAES v1.5 Key Encryption
RSA-OAEP	-2	RSAES OAEP w/ SHA-256
A128KW	-3	AES Key Wrap w/ 128-bit key
A192KW	-4	AES Key Wrap w/ 192-bit key
A256KW	-5	AES Key Wrap w/ 256-bit key
dir	-6	Direct use of CEK
ECDH-ES	*	ECDH ES w/ Concat KDF as CEK
ECDH-ES+A128KW	*	ECDH ES w/ Concat KDF and AES Key wrap w/ 128 bit key
ECDH-ES+A192KW	*	ECDH ES w/ Concat KDF and AES Key wrap w/ 192 bit key
ECDH-ES+A256KW	*	ECDH ES w/ Concat KDF and AES Key wrap w/ 256 bit key
A128GCMKW	*	AES GCM Key Wrap w/ 128 bit key
A192GCMKW	*	AES GCM Key Wrap w/ 192 bit key
A256GCMKW	*	AES GCM Key Wrap w/ 256 bit key
PBES2-HS256+A128KW	*	PBES2 w/ HMAC SHA-256 and AES Key wrap w/ 128 bit key
PBES2-HS384+A192KW	*	PBES2 w/ HMAC SHA-384 and AES Key wrap w/ 192 bit key
PBES2-HS512+A256KW	*	PBES2 w/ HMAC SHA-512 and AES Key wrap w/ 256 bit key
A128GCM	1	AES-GCM mode w/ 128-bit key
A192GCM	2	AES-GCM mode w/ 192-bit key

```
   | A256GCM             | 3     | AES-GCM mode w/ 256-bit key          |
   +--------------------+-------+--------------------------------------+
```

Appendix F. COSE General Values

```
            +------+--------+------------------------+
            | name | number | description            |
            +------+--------+------------------------+
            | EC   | 1      | Elliptic Curve key Type |
            |      |        |                        |
            | RSA  | 2      | RSA Key type           |
            |      |        |                        |
            | oct  | 3      | Octet Key type         |
            |      |        |                        |
            | P256 | 4      | EC Curve P256 (NIST)   |
            |      |        |                        |
            | P521 | 5      | EC Curve P521 (NIST)   |
            +------+--------+------------------------+
```

Appendix G. COSE Key Map Labels

   This table contains a list of all of the parameters defined for keys
   that were defined by the JOSE document set. In the table is the data
   value type to be used for CBOR as well as the integer value that can
   be used as a replacement for the name in order to further decrease
   the size of the sent item.

name	label	CBOR type	registry	description
kty	1	tstr / int	COSE General Values	Identification of the key type
use	*	tstr		deprecated - don't use
key_ops	*	[* tstr]		
alg	3	tstr / int	COSE Algorithm Values	Key usage restriction to this algorithm
kid	2	bstr		Key Identification value - match to kid in message
x5u	*	tstr		
x5c	*	bstr*		
x5t	*	bstr		
x5t#S256	*	bstr		

```
;key_keys
kty=1
key_kid=2
key_alg=3
```

Appendix H. COSE Key Parameter Labels

   This table contains a list of all of the parameters that were defined
   by the JOSE document set for a specific key type. In the table is
   the data value type to be used for CBOR as well as the integer value
   that can be used as a replacement for the name in order to further
   decrease the size of the sent item. Parameters dealing with keys

key type	name	label	CBOR type	registry	description
EC	crv	-1	int / tstr	Pull from general value registry	
EC	x	-2	bstr		
EC	y	-3	bstr		
EC	d	-4	bstr		
RSA	e	-1	bstr		
RSA	n	-2	bstr		
RSA	d	-3	bstr		
RSA	p	-4	bstr		
RSA	q	-5	bstr		
RSA	dp	-6	bstr		
RSA	dq	-7	bstr		
RSA	qi	-8	bstr		
RSA	oth	-9	bstr		
RSA	r	-10	bstr		
RSA	t	-11	bstr		
oct	k	-1	bstr		

Editorial Comments

[CREF1] JLS: Need to check this list for correctness before publishing.

[CREF2] JLS: I have moved msg_type into the individual structures.
       However, they would not be necessary in the cases where a) the
       security service is known and b) security libraries can setup to
       take individual structures. Should they be moved back to just
       appearing if used in a COSE_MSG rather than on the individual
       structure?

[CREF3] JLS: Should we create an IANA registries for the values of
        msg_type?

[CREF4] JLS: OPEN ISSUE

[CREF5] JLS: A completest version of this grammar would list the options
        available in the protected and unprotected headers. Do we want
        to head that direction?

[CREF6] JLS: Need to figure out how we are going to go about creating
        this registry -or are we going to modify the current mime-
        content table?

[CREF7] Ilari: I don't follow/understand this text

[CREF8] JLS: Finish the registration process.

[CREF9] JLS: Should we register both or just the cose+cbor one?

Author's Address

   Jim Schaad
   August Cellars

   Email: ietf@augustcellars.com