Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 12, 2015

J. Schaad
August Cellars

June 10, 2015

CBOR Encoded Message Syntax
draft-schaad-cose-msg-00

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. The work in this document is derived in part from the JSON web security documents using the same parameters and algorithm identifiers as they do.

Table of Contents

1. Introduction

The JOSE working group produced a set of documents 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 taking into account:

- 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 sting.

- The authors 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 authors to be superior or simpler, replaced.

1.1. Design changes from JOSE

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

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

- 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.

- MAC messages are separated from signed messages.

- MAC messages have the ability to do key management on the MAC key.

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

- Remove the authentiction tag for encryption algorithms as a separate item.

- 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 compenstate for this.

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].

2. The COSE_MSG structure

The COSE_MSG structure is a top level CBOR object which corresponds to the DataContent type in [RFC5652]. This structure allows for a top level message to be sent which could be any of the different security services, where the security service is identified. The presence of this structure does not preclude a protocol to use one of the individual structures as a stand alone component.

```
COSE_MSG = {msg_type=>1, COSE_Sign} /
           {msg_type=>2, COSE_encrypt} /
           {msg_type=>3, COSE_mac}

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

The top level of each of the COSE message structures are encoded as arrays.
We use an integer to distingish bettwen the different security message types. By looking at the integer in the first element, one can determine which security message is being used and thus what the syntax is for the rest of the elements in the array.

Implementations SHOULD be prepared to find an integer in the location 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. Clients need to recognize that the set of values could be extended at a later date, but should not provide a security service based on guesses of what is there.

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 message structure 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 maps can be keyed by negative integers, unsigned integers and strings. The negative and unsigned integers are used for compactness of encoding. The value portion is dependent on the key definition. Both maps use the same set of key/value pairs. The integer key range 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 keys defined can be found in Appendix E.

Two buckets are provided for each layer:

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:

```
keys = int / tstr
header_map = {+ keys => any }

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

3.1. COSE Headers

TODO: Do we need to repeat definitions for all or just for some and refer to the JOSE documents?

TODO: Should we move table Appendix E to here or leave it as an appendix. Some what redundant if we document things in text.

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. (Source of text is [RFC5652].) More detailed information on multiple signature evaluation can be found in [RFC5752].

The CDDL grammar structure for a signature message is:

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

The keys in the COSE_Sign map are keyed by the values in Appendix D. While other keys can be present in the map, it is not generally a recommended practice. The other keys can be either of integer or string type, use of other types is strongly discouraged. See the note in {{CBOR-Canonical} about options for allowing or disallowing other keys.

The fields is the structure have the following semantics:

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. This fields in this map are typically keyed by Appendix E. Other keys can be used either as int or tstr values. Other types MUST NOT be present in the map as key values.

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. This fields int his map are typically keyed by Appendix E. Other keys can be used either as int or tstr values. Other types MUST NOT be present in the map as key values.

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.

The keys in the COSE_signature map are keyed by the values in Appendix D. While other keys can be present in the map, it is not generally a recommended practice. The other keys can be either of integer or string type, use of other types is strongly discouraged. See the note in {{CBOR-Canonical} about options for allowing or disallowing other keys.

The CDDL grammar structure for a signature is:

```
COSE_signature =  (
    ? protected => bstr,
    ? unprotected => header_map,
    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 be used.

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

3. Comput the hash value from the byte string.

4. Sign the hash

5. Place the signature value into the appropriate signature field.

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.

I don't follow/understand this text{:aeds}

The CDDL grammar structure for encryption is:

```
COSE_encrypt = (
   Headers,
   ? iv => bstr,
   ? aad => bstr,
   ? ciphertext => bstr,
   ? recipients => [+COSE_encrypt_a]
)

COSE_encrypt_a = {COSE_encrypt}
```

Description of the fields:

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.

iv contains the initialization vector (IV), or it's equivalent, if one is needed by the encryption algorithm.

aad contains additional authenticated data (aad) supplied by the application. This field contains information about the plain text data that is authenticated, but not encrypted.

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. The field can have one of two data types:

o An array of COSE_encrypt elements, one for each recipient.

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:

- The 'protected', 'iv', 'aad', 'ciphertext' and 'recipients' fields MUST be absent.

- At a minimum, the 'unprotected' field SHOULD contain the 'alg' parameter as well as 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:

- The 'protected', 'aad', and 'recipients' fields MUST be absent.

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

- At a minimum, the 'unprotected' field SHOULD contain the 'alg' parameter as well as a parameter identifying the shared secret.

- Use of the 'iv' field will depend on the key wrap algorithm.

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:

- The 'protected', 'aad', and 'iv' fields MUST be absent.
- The plain text to be encrypted is the key from next layer down (usually the content layer).
- At a minimum, the 'unprotected' field SHOULD contain the 'alg' parameter as well as 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:

- The 'protected', 'aad', and 'iv' fields MUST be absent.
- At a minimum, the 'unprotected' field SHOULD contain the 'alg' parameter as well as a parameter identifying the asymmetric key.
- 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:

- The 'protected', 'aad', and 'iv' fields MUST be absent.
- The plain text to be encrypted is the key from next layer down (usually the content layer).
- At a minimum, the 'unprotected' field SHOULD 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,
   aad => bstr
]
```

1. If there is protected data, CBOR encode the map to a byte string and place in the protected field of the Enc_structure and the COSE_Encrypt structure.

2. Copy the 'aad' field from the COSE_Encrypt structure to the Enc_Structure.

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

4. Encrypt the plain text and place it in the 'ciphertext' field. The AAD value is passed in as part of the encryption process.

5. For recipient 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 the 'aad' field is absent.

3. Encrypt the plain text and place in the 'ciphertext' field.

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 = (
   Headers,
   ? payload => bstr,
   tag => bstr,
   ? recipients => [+COSE_encrypt_a]
)
```

Field descriptions:

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

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

4. 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 prefered 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:

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

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

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

  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 resrict 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 the Sig_structure, the Enc_structure, and the MAC_structure.

- 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.

- All parsers used SHOULD fail on both parsing and generation if the same key is used twice in a map.

While it is permitted to have key values other than those specified in this document in the outer maps (COSE_Sign, COSE_Signature, COSE_encrypt, COSE_recipient and COSE_mac), doing so is not encouraged. Applications should make a determination if it will be

permitted for that application. In general, any needed new fields can be accomadated by the introduction of new header fields to be carried in the protected or unprotected fields. Applications that need to have new fields in these maps should consider getting new message types assigned for these usages. Without this change, old applications will not see and process the new fields.

9. IANA Considerations

9.1. CBOR Tag assignment

It is requested that IANA assign a new tag from the "Concise Binary Object Represetion (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 Parameter Table

9.3. COSE Header Key Table

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

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.

key This is the value used for the key. The key can be either an integer or a string. Registration in the table is based on the value of the key 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 Key" registry. Integer values beyond -65536 are marked as private use.

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

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 E. 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 Key Table

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

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, key pair MUST be unique.

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

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

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 F. 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:

key The value to be used to identify this algorithm. Algorithm keys MUST be unique. The key 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 Key" 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 G. 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 entitied "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.

key The value to be used to identify this algorithm. Algorithm keys MUST be unique. The key 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 algoirthm delegated to the "COSE Key Parameter Key" 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 bythe values in Appendix I. 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.

key The key is to be unqiue 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 bythe values in Appendix J. The specification column for all of these entries will be this document.

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 direcct 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.

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

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.

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:

- 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].

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

- 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:

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

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

- 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.

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 albe 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 manangment 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'82C136D2C8CB27356635FAFE6F2E1AB2BC23FA706A33357DB017EE51710EEDE5',
 9: [
  {3: {1: "ECDH-SS", 5: "meriadoc.brandybuck@buckland.example",
       "spk": {"kid": "peregrin.took@tuckborough.example"},
       "apu": h'4D8553E7E74F3C6A3A9DD3EF286A8195CBF8A23D19558CCFEC7D34
       B824F42D92BD06BD2C7F0271F0214E141FB779AE2856ABF585A58368B017E7F2A
       9E5CE4DB5'}}]}
```

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 exmple uses AES-128-MAC trucated to 64-bits as the digest algorithm. It uses AES-256 Key wrap for the key manangment algorithm wrapping the 128-bit key used for the digest algorthm.

```
{1: 3, 2: h'A1016E4145532D3132382D4D41432D3634',
 4: h'546869732069732074686520636F6E74656E742E',
 10: h'A61AE6CFB7CABCC9', 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 manangment techniques applied:

- An ephemeral static ECDH key agrement operation using AES-128 key wrap on the digest key.
- Key transport using RSA-OAEP with SHA-256 for the hash and the mfg function operations.
- AES 256-bit Key wrap using a pre-shared secret.

```
{1: 3, 2: h'A10104', 4: h'546869732069732074686520636F6E74656E742E',
 10: h'051FA3288A39AC726B4FAE79A4B93FB17D8DC3F6E666247EE7AD40CE1665FCDE',
 9: [
   {3: {1: "ECDH-ES+A128KW",
        5: h'62696C626F2E62616767696E7340686F626269746F6E2E6578616D706C65',
        4: {1: 1, -1: 5, -2: h'43B12669ACAC3FD27898FFBA0BCD2E6C366D53BC4DB
            71F909A759304ACFB5E18CDC7BA0B13FF8C7636271A6924B1AC63C02688075
            B55EF2D613574E7DC242F79C3',
            -3: h'812DD694F4EF32B11014D74010A954689C6B6E8785B333D1AB44F22B
            9D1091AE8FC8AE40B687E5CFBE7EE6F8B47918A07BB04E9F5B1A51A334A16B
            C09777434113'}},
    4: h'1B120C848C7F2F8943E402CBDBDB58EFB281753AF4169C70D0126C0D164362771
         60821790EF4FE3F'},
   {3: {1: -2, 5: h'62696C626F2E62616767696E7340686F626269746F6E2E6578616D
        706C65'},
    4: h'46C4F88069B650909A891E84013614CD58A3668F88FA18F3852940A20B3509859
         1D3AACF91C125A2595CDA7BEE75A490579F0E2F20FD6BC956623BFDE3029C318
        F82C426DAC3463B261C981AB18B72FE9409412E5C7F2D8F2B5ABAF780DF6A282D
        B033B3A863FA957408B81741878F466DCC437006CA21407181A016CA608CA8208
        BD3C5A1DDC828531E30B89A67EC6BB97B0C3C3C92036C0CB84AA0F0CE8C3E4A21
        5D173BFA668F116CA9F1177505AFB7629A9B0B5E096E81D37900E06F561A32B6B
        C993FC6D0CB5D4BB81B74E6FFB0958DAC7227C2EB8856303D989F93B4A0518307
        06A4C44E8314EC846022EAB727E16ADA628F12EE7978855550249CCB58'},
   {3: {1: -5, 5: "018c0ae5-4d9b-471b-bfd6-eef314bc7037"},
        4: h'0B2C7CFCE04E98276342D6476A7723C090DFDD15F9A518E7736549E99837
        0695E6D6A83B4AE507BB'}]}
```

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', 7: h'C9CF4DF2FE6C632BF7886413',
 4: h'45FCE2814311024D3A479E7D3EED063850F3F0B94EE043BAFDFA14636E632CF6
      75AF2DAE',
  9: [{3: {1: "ECDH-ES", 5: "meriadoc.brandybuck@buckland.example",
       4: {1: 1, -1: 4, -2: h'98F50A4FF6C05861C8860D13A638EA56C3F5AD75
           90BBFBF054E1C7B4D91D6280',
          -3: h'F01400B089867804B8E9FC96C3932161F1934F4223069170D924B7
           E03BF822BB'}}}]}
```

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'546869732069732074686520636F6E74656E742E',
 5: [{2: h'A2016550533383405781E62696C626F2E62616767696E7340686F626
          269746F6E2E6578616D706C65',
       6: h'4D645B5FF17BCDAD7EB29ABA0EBBFFA747E72767714F26EDBC5B4C1D2
         1CBE799B71388CCC73BDB25C4443D0EA2226B774A5B4815ABA82233B33DA
         4C3958D08285384A854A8F7F8FA9635A1A63BAB2A5D8CF45939A7FA2D95C
         C827EF94EF85276611B957B402BD1756D952597751C7AF5D26023012D3DC
         BFD785F9C0BE57F60719EFB0D2F9280A8D2B18D142F76942D007B4E24087
         DA4BE8F793B646D7B03A86C12731A8EDB36A95DFE6C281B58388380354A2
         94CC21DBC1C1EEE2DB35293AD406F50283874475B9A7E22920BD79B3D055
         214EE1C9D941F125548B9F23A87DCC26CBBEFD0919CF6F89E192A78130AC
         018D1921EF5B4D0A47659E9CBC1CE58ED26'}]}
```

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'1FD44A2BA1A8A0A664024E7E2AFD1D1D1159460E3C03B9BE8C8F60639CE
        614F59AF33108B65BBDEF3C330FB97E335DA11EEA9B6CBD7E7908FB8B5F61D
        FEB76EC6ED6A62BD9F3D338E373E1903CE2D5D3BD20086BBCA82A6F424E9F4
        1591BD6261835A74F0C0425E88666D530B72ADC1E33C10DC1D0361922B6ADC
        685B76E5CEA79FACA7C4CB66B1379B3F852A5ACE79A5812C6EE1CD3CC7CC88
        F2C9D30FF89D3BD0DE2D0C9355E9712B1BA8AB2F2B065BE0A0D93BFFA27DA0
        2221865A2B16093D92F71F9864D92C87057AE591334DB4CF881ECBEC2AC727
        77D9C88871C10733D65566B35FBFA6BAB54078C1C73AE8758196221FB2814E
        C283A95D191FB80D616'},
     {3: {1: -9, 5: "bilbo.baggins@hobbiton.example"},
      6: h'32247A4FD1CA2B69EEEB48CE65D07F2089D79271BB94847F8628DADB7AF
           FC1A34C24D10DB3C5E0D00BD9CB3BFB9666BAD6E9752564D35C5CCE375B
           A44E2FF33336008D8E07484041DBEFB179EBFFA5455E05D6B24E22DAECF
           0D76AD041A13A9DD7E3DAED7F6B09F1831092FFC5CB8BFE7DBF5E047858
           02A4CB741395F81E76A3A8AD61'}]}
```

Appendix D. Top Level Parameter Table

This table contains the list of all key values that can ocur in the COSE_Sign, COSE_signature, COSE_Encrypt, and COSE_MAC structures.

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
iv	7	For COSE_encrypt only
aad	8	For COSE_encrypt 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

```
; message_keys
msg_type=1
protected=2
unprotected=3
payload=4
signatures=5
signature=6
iv=7
aad=8
ciphertext=4
recipients=9
tag=10
```

M00TODO: 1. There is no equivalent to this table in JOSE so we need to get a name for the table and registration rules. 2. Initial registration rules: Number may be a positive or a negative value. Values in the range of -24 to 24 are Standards action required. Values in the range of -256 to -25 and 25 to 255 are specification required with expert review. Values from 256 to 512 are designated for private use. All other values are reserved.

Appendix E. COSE Header Key Registry

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	key	value	registry	description
alg	1	int / tstr	COSE Algorithm Registry	Integers are taken from table Appendix G
crit	2	[+ (tstr/int)]	COSE Header Key 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
epk	4	COSE_Key		contains a COSE key not a JWK key
jku	*	tstr		URL to COSE key object
jwk	*	COSE_Key		contains a COSE key not a JWK key
kid	*	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 G

Appendix F. COSE Header Algorithm Key Table

name	algorithm	key	CBOR type	description
apu	ECDH	-1	bstr	
apv	ECDH	-2	bstr	
iv	A128GCMKW, A192GCMKW, A256GCMKW	-1	bstr	
iv	A128GCM, A192GCM, A256GCM	-1	bstr	
p2c	PBE	-1	int	
p2s	PBE	-2	bstr	

Appendix G. COSE Algorithm Name Values

This table contains all of the defined algorithms for COSE.

name	key	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 H. 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 I. COSE Key Map Keys

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	key	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 J. COSE Key Parameter Keys

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	key	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		

Author's Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com