JOSE Working Group
Internet-Draft
Intended status: Informational
Expires: April 30, 2015

C. Bormann
Universitaet Bremen TZI

October 27, 2014

Constrained Object Signing and Encryption (COSE)
draft-bormann-jose-cose-00

Abstract

COSE provides services similar to JSON Web Signature (JWS), JSON Web Encryption (JWE), and JSON Web Key (JWK), making use of JSON Web Algorithms (JWA), for data encoded in the Concise Binary Object Representation (CBOR).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Constrained nodes and networks of constrained nodes [RFC7228] pose some specific requirements on data representation that may make it difficult to apply existing object security standards to this space.

In constrained node networks, message payloads are often small (by nature of the data exchanged), and both transmission systems (e.g., [RFC4944]) and application protocols (e.g., [RFC7252]) are optimized for these small interchanges. As a result, fixed-size overheads introduced by security protocols may be much more detrimental than in a traditional Web environment. Transmission/reception of messages requires power, turning a system that on average might consume ~100 [micro]W into a 50 mW consumer while communicating. This is a strong incentive to keep message sizes reasonably small. It is not often possible to rely on compression to achieve this, as compression requires CPU power, RAM, and code space, which all are rather constrained in these environments; compression also works better for larger messages.

Handling messages requires RAM, the total available size of which on a constrained node may be on the order of 10 KiB (note that apart from security, most of this RAM is already needed for operating system, network stack, sensor management, application processing, etc.). Protocols that require copying data, or, worse, re-encoding and escape processing, can double or triple those RAM requirements.

All the processing that is to be performed in a constrained node requires code space in Flash, the total available size of which on a constrained node may be on the order of 100 KiB (with the same note applying as above). This leads to a strong requirement to minimize

code complexity, and in particular to avoid having to implement multiple different ways to do the same thing.

Still, security is not optional.

1.1. Objectives

The JOSE set of specifications provides an attractive set of functions for the constrained space, even if its breadth of optional functionality may go beyond what is required there. The present specification aims to make use of the substantial amount of work that went into making JOSE such a comprehensive specification.

JOSE makes use of JSON [RFC7159], a text based data representation format. For applications in constrained nodes, the Concise Binary Object Representation format (CBOR) provides a more compact representation that is still largely based on the same principles.

By using CBOR, the present specification can:

- avoid the use of base64 coding of binary data. Base64 coding causes message expansion, which is detrimental to energy requirements. In an implementation, it also causes the requirement for creating copies of some data, which increases RAM requirements.

- avoid JSON-encoding of data. Again, this causes some message expansion, requires creating copies for escape processing, but also requires considerable code size, including for binary-to-decimal conversion.

- potentially make use of CBOR's capability to minimize strings by enumerating frequently occurring member names. This again helps to reduce message sizes, but also can save some code space. (This is a secondary, but useful objective.)

1.2. COSE

COSE provides services similar to JSON Web Signature (JWS) [I-D.ietf-jose-json-web-signature], JSON Web Encryption (JWE) [I-D.ietf-jose-json-web-encryption], and JSON Web Key (JWK) [I-D.ietf-jose-json-web-key], making use of JSON Web Algorithms (JWA) [I-D.ietf-jose-json-web-algorithms], in conjunction with data encoding in the Concise Binary Object Representation (CBOR) [RFC7049].

1.3. 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 RFC 2119 [RFC2119].

The term "byte" is used in its now customary sense as a synonym for "octet".

2. Specification

Presently, we believe the entire specification of COSE can be reduced to the following:

COSE is exactly like JOSE, except that:

- o each use of JSON is replaced by the equivalent use of CBOR;
- o base64-encoding is never done:
  - * where the output of the base64url function was to be used as a JSON string, instead the input to the base64url function is represented as a byte string in CBOR
  - * where the output of the base64url function was to be used as an input to a cryptographic algorithm, instead the input is used directly
  - * where the output of the base64url function was to be joined by ASCII dots (".") with other such outputs, CBOR encoding of an array built from the inputs, each represented as a byte string, is used.
- o (probably:) certain member names ("alg"...) are replaced by a number of predefined numeric replacements only in the key positions of maps (JSON objects) defined by JOSE. Only the names most likely to be frequently occurring in constrained node networks are entered into a static table to be defined in this specification. (There is no future extension planned for this table.)

3. Examples

(TBD, to cover large parts of [I-D.ietf-jose-cookbook]).

4. IANA considerations

(TBD)

5. Security considerations

(TBD)

6. Acknowledgments

This document obviously owes a lot to the work of the entire JOSE working group, including the feedback during an initial presentation at IETF 90 [1]. Richard Barnes' Python implementation of JOSE was instrumental in confirming the feasibility of the COSE approach.

7. References

7.1. Normative References

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-36 (work in progress), October 2014.

[I-D.ietf-jose-json-web-encryption]
Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-36 (work in progress), October 2014.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-36 (work in progress), October 2014.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-36 (work in progress), October 2014.

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

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

7.2. Informative References

[I-D.ietf-jose-cookbook]
Miller, M., "Examples of Protecting Content using JavaScript Object Signing and Encryption (JOSE)", draft-ietf-jose-cookbook-05 (work in progress), October 2014.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

7.3. URIs

[1] http://www.ietf.org/proceedings/90/slides/slides-90-jose-2.pdf

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 8, 2015

J. Schaad
August Cellars
February 04, 2015

CBOR Encoded Message Syntax
draft-schaad-cose-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.

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 August 8, 2015.

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:

- 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 sting.

- o 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

- 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 Switch from using a map to using an array at the message level. While this change means that it is no longer possible to add new named parameters to the top level message, it also means that there is not a need to define how older implementations are defined to behave when new fields are present. Most of the reasons that a new field would need to be defined are adequately addressed by defining a new parameter instead.

- 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 key.

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

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

1.2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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 a modified version of the CBOR data definition language (CDDL) defined in [I-D.greevenbosch-appsawg-cbor-cddl]. The differences between the defined grammar and the one we used are mostly self explanatory. The biggest difference being the addition of the choice operator '|'. Additionally, note the use of the null value which is used to occupy a location in an array but to mark that the element is not present.

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 : uint;
  msg_content : COSE_Sign | COSE_encrypt | COSE_mac;
}
```

This structure is encoded as an array by CBOR. Descriptions of the fields:

msg_type indicates which of the security structures is in this block.

msg_content contains the top level fields for the security service provided. The type in this field is based on the value of the field msg_type.

msg_type 1 is used for COSE_sign

msg_type 2 is used for COSE_encrypt

msg_type 3 is used for COSE_mac

3. 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 : {
    protected : bstr / null;
    unprotected : map(tstr, .) / null;
    payload : bstr / null;
    signatures: COSE_signature_a* / COSE_signature;
}
```

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.

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.

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 either a single signature or an array of signature values.
A single signature value can be represented using either data type. Implementations MUST be able to parse both data types.

The CDDL grammar structure for a signature is:

```
COSE_signature :  {
    protected : bstr / null;
    unprotected : map(tstr, .) / null;
    signature : bstr;
}
*COSE_signature_a : COSE_signature;
```

The fields is 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 / null;
    sign_protected : bstr / null;
    payload : bstr;
}
```

How to compute a signature:

1. Create a Sig_structure object and populate it with the appropriate fields.

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.

4. 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 [I-D.ietf-jose-json-web-encryption] 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.

The CDDL grammar structure for encryption is:

```
COSE_encrypt {
  protected : bstr | null;   # Contains map(tstr, .)
  unprotected : map(tstr, .) | null;
  iv : bstr | null;
  aad : bstr | null;
  ciphertext : bstr | null;
  recipients : COSE_encrypt_a* | COSE_encrypt | null;
}

* 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' if present and the value 'null' if there is no data. 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. If there are no field, then the value 'null' is used. 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. If there is no IV, then the value 'null' is used.

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. If the application does not provide this data, the value 'null' is used.

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 value 'null' is encoded here.

recipients contains the recipient information. The field can have one of three data types:

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

- o A single COSE_encrypt element, encoded as an extension to the containing COSE_encrypt element, for a single recipient. Single recipients can be encoded either this way or as a single array element.

- o A 'null' value if there are no recipients.

4.1. 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 [I-D.ietf-jose-json-web-key]. 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 [I-D.ietf-jose-json-web-algorithms] were extended to allow for authenticated data. In that event the use of the 'protected' field, which is current forbidden below, would be permitted.

4.1.1. Direct Encryption

In direct encryption mode, a shared secret between the sender and the recipient is used as the CEK. For direct encryption mode, no recipient structure is built. All of the information about the key is placed in either the protected or unprotected fields at the content level. 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', 'iv', 'aad', 'ciphertext' and 'recipients' fields MUST be null.

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

4.1.2. 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 null.
- 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.

4.1.3. 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 all use the 'null' value.
- 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.

4.1.4. Direct Key Agreement

Direct Key Agreement derives the CEK from the shared secret computed by the key agreement operation. For Direct Key Agreement, no recipient structure is built. All of the information about the key and key agreement process is placed in either the 'protected' or 'unprotected' fields at the content level.

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 all use the 'null' value.
- 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.

4.1.5. 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 all use the 'null' value.
- 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.

4.2. 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 | null;
   aad : bstr | null;
}
```

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 7 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.

4.3. Encryption algorithm for AE algorithms

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

2. Verify that the 'aad' field is empty.

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

5. 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 :  {
   protected : bstr | null;
   unprotected" : map(tstr, .) | null;
   payload : bstr;
   tag : bstr;
   recipients : COSE_encrypt_a* | COSE_encrypt | null;
}
```

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 | null;
   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.

6. 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 [I-D.ietf-jose-json-web-key].

```
COSE_Key : map {
    "kty" : tstr;
    "use" : tstr;
    "key_ops" : tstr*;
    "alg" : tstr;
    "kid" : tstr;
}

*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 [I-D.ietf-jose-json-web-key] or [I-D.ietf-jose-json-web-algorithms] 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 [I-D.ietf-jose-json-web-key] 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.

7. 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:

- 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 key is used twice in a map.

8. IANA Considerations

There are IANA considerations to be filled in.

9. 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.

10. References

10.1. Normative References

[I-D.greevenbosch-appsawg-cbor-cddl]
Greevenbosch, B., Sun, R., and C. Vigano, "CBOR data definition language: a notational convention to express CBOR data structures.", draft-greevenbosch-appsawg-cbor-cddl-04 (work in progress), December 2014.

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-40 (work in progress), January 2015.

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

10.2. Informative References

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

[I-D.ietf-jose-json-web-encryption]
Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-40 (work in progress), January 2015.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-41 (work in progress), January 2015.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 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.

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.

PKCS 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.

Appendix C. 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.

Encoded in CBOR - 118 bytes, content is 14 bytes long

```
[
  2,
  null,
  {
    "alg": "HS256"
  },
  h'436f6e74656e7420537472696e67',
  h'78956d858ee6c026ac630063627a4ce98d3003bc68e7c1e53b5b468331b69f93',
  null,
  {
    "alg": "dir",
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
  },
  null,
  null,
  null
]
```

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.

Encoded in CBOR - 162 bytes, content is 14 bytes long

```
[
  2,
  null,
  {
    "alg": "HS256"
  },
  h'436f6e74656e7420537472696e67',
  h'2ee486376b8b2a61fe526589ceb456e20919a68ebc0458431ef3e13ffe7b
    f698',
  null,
  {
    "alg": "A128KW",
    "kid": "77c7e2b8-6e13-45cf-8672-617b5b45243a"
  },
  null,
  h'4f6e9e6a3e43b79561ef602a2a9e629a437e8df90a7ff361acbdb1076c95
    5d0f25c660a67aee1bdf',
  null
]
```

C.3. 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,
  null,
  {"alg": "A128GCM"},
  h'656d6a73ccf1b35fb99044e1',
  h'd7b27b67a81b212ee513b148454fe2d571d51bb679239769f5d2299bb96b',
  null,
  {
    "alg": "ECDH-ES",
    "epk": {
      "kty": "EC",
      "crv": "P-256",
      "x": h'00b81ff1de0eeba27613027526d83b5f4cbffaca433488e3805
             e7a75c43bd1b966',
      "y": h'00d142a334ac8790dc821abe9362434daeb00c1b8b076843e51
             a4a4717b30c54ce'},
      "kid": "meriadoc.brandybuck@buckland.example"
    }
  },
  null,
  null,
  null
]
```

C.4. 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.

```
[
  0,
  null,
  null,
  h'436f6e74656e7420537472696e67',
  null,
  {
    "kid": "bilbo.baggins@hobbiton.example",
    "alg": "PS256"
  },
   h'5afe80ec9f208b4719a3bd688c803a3154b1ff25af86e054173ad6ddf71
     ba77a4a2b793beed077a4e1a8a69ac1277c457f636691cb4a7d3dc67b47
     ec84c067076b720236bae498bdb21deebbc0a0f525f9a24b336d51e2b3e
     ffd67df3e051405a3599aed83b8a8e94e4194dded2f661e5e6894825779
     b79b463bd4f477f33356cf8aecfa8a543344d2620145be8a72a712f9854
     57040140176164c77cdae7cc480ae4357683cce79b97ddb10f390862a24
     2aae1aa391cc730b1631f020874a8a6efc77b08f027323e2c4ae85eeb3e
     5dc715e0e2fa8aec63fb828d7a2c45e361e249117bd8b41e1e12388412d
     8ce3809c9a2172afda5ca7c5839896825da66a50'
]
```

C.5. 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

```
[
  0,
  null,
  null,
  h'436f6e74656e7420537472696e67',
  [
    [
      null,
      {
        "kid": "bilbo.baggins@hobbiton.example",
        "alg": "PS256"
      },
       h'5afe80ec9f208b4719a3bd688c803a3154b1ff25af86e054173ad6d
         df71ba77a4a2b793beed077a4e1a8a69ac1277c457f636691cb4a7d
         3dc67b47ec84c067076b720236bae498bdb21deebbc0a0f525f9a24
         b336d51e2b3effd67df3e051405a3599aed83b8a8e94e4194dded2f
         661e5e6894825779b79b463bd4f477f33356cf8aecfa8a543344d26
         20145be8a72a712f985457040140176164c77cdae7cc480ae435768
         3cce79b97ddb10f390862a242aae1aa391cc730b1631f020874a8a6
         efc77b08f027323e2c4ae85eeb3e5dc715e0e2fa8aec63fb828d7a2
         c45e361e249117bd8b41e1e12388412d8ce3809c9a2172afda5ca7c
         5839896825da66a50'
    ],
    [
      null,
      {
        "kid": "bilbo.baggins@hobbiton.example",
        "alg": "ES512"
      },
      h'00e9769c05afb2d93baf5a0c2cace1747b5091f50596831911c67ebf
        76f4220adb53698fe7831000d526887893d67de05ead1bbe378ce9e9
        731bda4cd37f53dcf8d40186c46d872795b566682c113cc9d5bf5a8c
        5321fd50a003237115decf0cb8b09e5c3cb50bc2203af45bebd51e6c
        4d0ec51170d5b9ac1b21a2017a50d7c15b6de8f9'
    ]
  ]
]
```

Appendix D. Processing Parameter Table

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	number	CBOR type	comments
alg	*	tstr	presence is required
apu	*	bstr	
apv	*	bstr	
crit	*	tstr*	
cty	*	tstr	
enc	*		use alg instead
epk	*	map	contains a COSE key not a JWK key
iv	*		use field in array instead
jku	*	tstr	
jwk	*	map	contains a COSE key not a JWK key
kid	*	tstr	
p2c	*	int	
p2s	*	bstr	
tag	*		tag is included in the cipher text
typ	*		use cty for the content type, no concept of a different wrapper type
x5c	*	bstr*	
x5t	*	bstr	
x5t#S256	*	bstr	
x5u	*	tstr	
zip	*	tstr	only used at content level

Appendix E. Key Parameter Tables

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	number	CBOR type
kty	*	tstr
use	*	tstr
key_ops	*	tstr*
alg	*	tstr
kid	*	tstr
x5u	*	tstr
x5c	*	bstr*
x5t	*	bstr
xt5#S256	*	bstr

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	number	CBOR type
EC	d	*	bstr
EC	x	*	bstr
EC	y	*	bstr
RSA	e	*	bstr
RSA	n	*	bstr
RSA	d	*	bstr
RSA	p	*	bstr
RSA	q	*	bstr
RSA	dp	*	bstr
RSA	dq	*	bstr
RSA	qi	*	bstr
RSA	oth	*	bstr
RSA	r	*	bstr
RSA	t	*	bstr
oct	k	*	bstr

Author's Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com