COSE Working Group
Internet-Draft
Intended status: Informational
Expires: January 6, 2016

J. Schaad
August Cellars
July 5, 2015

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

Abstract

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

Contributing to this document

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at [1]. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the COSE mailing list.

Status of This Memo

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 January 6, 2016.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

1.1. Design changes from JOSE

- Define a top level message structure so that encrypted, signed and MACed 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 authentication key.
- Use binary encodings for binary data rather than base64url encodings.
- Combine the authentication tag for encryption algorithms with the ciphertext.
- Remove the flattened mode of encoding. Forcing the use of an array of recipients at all times forces the message size to be two bytes larger, but one gets a corresponding decrease in the implementation size that should compensate for this. [CREF1]

1.2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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

1.3. CBOR Grammar

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

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

```
start = COSE_MSG
```

The collected CDDL can be extracted from the XML version of this document via the following XPath expression below. (Depending on the XPath evaluator one is using, it may be necessary to deal with > as an entity.)

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

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

1.4. CBOR Related Terminology

In JSON, maps are called objects and only have one kind of map key: a string. In COSE, we use both strings and integers (both negative and non-negative integers) as map keys, as well as data items to identify specific choices. The integers (both positive and negative) are used for compactness of encoding and easy comparison. (Generally, in this document the value zero is going to be reserved and not used.) Since the work "key" is mainly used in its other meaning, as a cryptographic key, we use the term "label" for this usage of either an integer or a string to identify map keys and choice data items.

```
label = int / tstr
```

1.5. Mandatory to Implement Algorithms

One of the standard issues that is specified in IETF cryptographic algorithms is a requirement that a standard specify a set of minimal algorithms that are required to be implemented. This is done to promote interoperability as it provides a minimal set of algorithms that all devices can be sure will exist at both ends. However, we have elected not to specify a set of mandatory algorithms in this document.

It is expected that COSE is going to be used in a wide variety of applications and on a wide variety of devices. Many of the constrained devices are going to be setup to used a small fixed set

of algorithms, and this set of algorithms may not match those available on a device. We therefore have deferred to the application protocols the decision of what to specify for mandatory algorithms.

Since the set of algorithms in an environment of constrained devices may depend on what the set of devices are and how long they have been in operation, we want to highlight that application protocols will need to specify some type of discovery method of algorithm capabilities. The discovery method may be as simple as requiring preconfiguration of the set of algorithms to providing a discovery method built into the protocol. S/MIME provided a number of different ways to approach the problem:

- Advertising in the message (S/MIME capabilities)
- Advertising in the certificate (capabilities extension)
- Minimum requirements for the S/MIME which have been updated over time

2. The COSE_MSG structure

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

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

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

Each of the top-level security objects use a CBOR map as the base structure. Items in the map at the top level are identified by a label. This document defines a number of labels in the IANA "COSE Object Labels Registry" (defined in Section 9.2).

The set of labels present in a security object is not restricted to those defined in this document. However, it is not recommended that

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

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

0 - Reserved.

1 - Signed Message.

2 - Encrypted Message

3 - Authenticated Message (MACed message)

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

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

The CDDL grammar that corresponds to the above is:

```
COSE_MSG = COSE_Sign /
    COSE_encrypt /
    COSE_mac

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

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

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

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

Table 1: COSE Map Labels

The CDDL grammar that provides the label values is:

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

3. Header Parameters

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

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

Two buckets are provided for each layer: [CREF5]

protected contains attributes about the layer that 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 that 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 that describes the two buckets is:

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

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

3.1. COSE Headers

The set of header fields defined in this document are:

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

crit This field is used to ensure that applications will take appropriate action based on the values found. The field is used to indicate which protected header labels an application that is processing a message is required to understand. This field uses the integer '2' for the label. The value is an array of COSE Header Labels. When present, this MUST be placed in the protected header bucket.

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

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

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

cty This field is used to indicate the content type of the data in the payload or ciphertext fields. The field uses the integer of '3' for the label. The value can be either an integer or a string. [CREF6] Integers are from the XXXXX[CREF7] IANA registry table. Strings are from the IANA 'mime-content types' registry. Applications SHOULD provide this field if the content structure is potentially ambiguous.

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

'kid' value, it may be required that all of the keys need to be checked to find the correct one. This field uses the integer value of '4' for the label. The value of field is the CBOR 'bstr' type. The internal structure of 'kid' is not defined and generally cannot be relied on by applications. Key identifier values are hints about which key to use, they are not directly a security critical field, for this reason they can normally be placed in the unprotected headers bucket.

nonce This field holds either a nonce or Initialization Vector value. This value can be used either as a counter value for a protocol or as an IV for an algorithm. TODO: Talk about zero extending the value in some cases.

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

name	label	value	registry	description
alg	1	int / tstr	COSE Algorithm Registry	Integers are taken from table --POINT TO REGISTRY--
crit	2	[+ label]	COSE Header Label Registry	integer values are from this table.
cty	3	tstr / int	media-types registry	Value is either a media-type or an integer from the media-type registry
jku	*	tstr		URL to COSE key object
jwk	*	COSE_Key		contains a COSE key not a JWK key
kid	4	bstr		key identifier
x5c	*	bstr*		X.509 Certificate Chain
x5t	*	bstr		SHA-1 thumbprint of key
x5t#S256	*	bstr		SHA-256 thumbprint of key
x5u	*	tstr		URL for X.509 certificate
zip	*	int / tstr		Integers are taken from the table --POINT TO REGISTRY--
nonce	5	bstr		Nonce or Initialization Vector (IV)

Table 2: Header Labels

OPEN ISSUES:

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

2. I am currently torn on the question "Should epk and iv/nonce be algorithm specific or generic headers?" They are really specific to an algorithm and can potentially be defined in different ways for different algorithms. As an example, it would make sense to defined nonce for CCM and GCM modes that can have the leading zero bytes stripped, while for other algorithms this might be undesirable.

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

4. Signing Structure

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

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

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

The CDDL grammar for a signature message is:

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

The fields is the structure have the following semantics:

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

protected contains attributes about the payload that 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 that is encoded to a byte stream. This field MUST NOT contain attributes about the signature, even if those attributes are common across multiple signatures. The labels in this map are typically taken from Table 2.

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

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

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

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

The CDDL grammar structure for a signature is:

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

The fields in the structure have the following semantics:

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

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

signature contains the computed signature value.

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

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

How to compute a signature:

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

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

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

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

How to verify a signature:

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

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

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

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

5. Encryption object

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

One of the byproducts of using the same technique for encrypting and encoding both the content and the keys using the various key management techniques, is a requirement that all of the key management techniques use an Authenticated Encryption (AE) algorithm. (For the purpose of this document we use a slightly loose definition of AE algorithms.) When encrypting the plain text, it is normal to use an Authenticated Encryption with Additional Data (AEAD) algorithm. For key management, either AE or AEAD algorithms can be used. See Appendix A for more details about the different types of algorithms. [CREF8]

The CDDL grammar structure for encryption is:

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

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

Description of the fields:

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

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

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

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

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

5.1. Key Management Methods

This section has moved. Still need to make some small comments here.

5.2. Encryption Algorithm for AEAD algorithms

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

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

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

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

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

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

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

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

   Other: The key is randomly generated.

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

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

5.3. Encryption algorithm for AE algorithms

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

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

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

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

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

Other: The key is randomly generated.

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

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

6. MAC objects

In this section we describe the structure and methods to be used when doing MAC authentication in COSE. JOSE used a variant of the signature structure for doing MAC operations and it is restricted to using a single pre-shared secret to do the authentication. [CREF9] 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 MACing the message can be validated by a key binding. (The binding of identity assumes that there are only two parties involved and you did not send the message yourself.)

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

Field descriptions:

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

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

tag contains the MAC value.

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

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

How to compute a MAC:

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

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

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

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

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

7. Key Structure

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

For COSE we use the same set of fields that were defined in [RFC7517]. [CREF10] [CREF11]

```
COSE_Key = {
    kty => tstr / int,
    ? key_ops => [+ tstr / int ],
    ? alg => tstr / int,
    ? kid => bstr,
    * label => 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.

7.1. COSE Key Map Labels

This document defines a set of common map elements for a COSE Key object. Table 3 provides a summary of the elements defined in this section. There are also a set of map elements that are defined for a specific key type.

name	label	CBOR type	registry	description
kty	1	tstr / int	COSE General Values	Identification of the key type
key_ops	4	[* (tstr/int)]		Restrict set of permissible operations
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		
use	*	tstr		deprecated - don’t use

Table 3: Key Map Labels

kty: This field is used to identify the family of keys for this structure, and thus the set of fields to be found.

alg: This field is used to restrict the algorithms that are to be used with this key. If this field is present in the key structure, the application MUST verify that this algorithm matches the algorithm for which the key is being used. If the algorthms do not match, then this key object MUST NOT be used to perform the cryptographic operation. Note that the same key can be in a different key structure with a different or no algorithm specified, however this is considered to be a poor security practice.

kid: This field is used to give an identifier for a key. The identifier is not structured and can be anything from a user

provided string to a value computed on the public portion of the key. This field is intended for matching against a 'kid' field in a message in order to filter down the set of keys that need to be checked.

key_ops: This field is defined to restrict the set of operations that a key is to be used for. The value of the field is an array of values from Table 4.

Only the 'kty' field MUST be present in a key object. All other members may be omitted if their behavior is not needed.

name	value	description
sign	1	The key is used to create signatures. Requires private key fields.
verify	2	The key is used for verification of signatures.
encrypt	3	The key is used for key transport encryption.
decrypt	4	The key is used for key transport decryption. Requires private key fields.
wrap key	5	The key is used for key wrapping.
unwrap key	6	The key is used for key unwrapping. Requires private key fields.
key agree	7	The key is used for key agreement.

Table 4: Key Operation Values

The following provides a CDDL fragment which duplicates the assignment labels from Table 3 and Table 4.

```
;key_labels
key_kty=1
key_kid=2
key_alg=3
key_ops=4

;key_ops values
key_ops_sign=1
key_ops_verify=2
key_ops_encrypt=3
key_ops_decrypt=4
key_ops_wrap=5
key_ops_unwrap=6
key_ops_agree=7
```

8. CBOR Encoder Restrictions

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

- 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 label is used twice as a key for the same map.

9. IANA Considerations

9.1. CBOR Tag assignment

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

Tag Value: TBD1

Data Item: COSE_Msg

Semantics: COSE security message.

9.2. COSE Object Labels Registry

It is requested that IANA create a new registry entitled "COSE Object Labels Registry". [CREF12]

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

9.3. COSE Header Label Table

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

The columns of the registry are:

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

label This is the value used for the label. The label can be either an integer or a string. Registration in the table is based on the value of the label requested. Integer values between 1 and 255 and strings of length 1 are designated as Standards Track Document required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values of greater than 65535 and strings of length greater than 2 are designated as first come first server. Integer values in the range -1 to -65536 are delegated to the "COSE Header Algorithm Label" registry. Integer values beyond -65536 are marked as private use.

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

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

description This contains a brief description of the header field.

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

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

Additionally, the value of 0 is to be marked as ’Reserved’.

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

9.4. COSE Header Algorithm Label Table

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

The columns of the registry are:

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

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

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

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

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

description This contains a brief description of the header field.

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

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

9.5. COSE Algorithm Registry

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

The columns of the registry are:

value The value to be used to identify this algorithm. Algorithm values MUST be unique. The value can be a positive integer, a negative integer or a string. Integer values between 0 and 255 and strings of length 1 are designated as Standards Track Document

required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values of greater than 65535 and strings of length greater than 2 are designated as first come first server. Integer values in the range -1 to -65536 are delegated to the "COSE Header Algorithm Label" registry. Integer values beyond -65536 are marked as private use.

description A short description of the algorithm.

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

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

9.6. COSE Key Map Registry

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

The columns of the registry are:

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

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

CBOR Type This field contains the CBOR type for the field

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

description This field contains a brief description for the field

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

This registry will be initially populated by the values in Section 7.1. The specification column for all of these entries will be this document.

9.7. COSE Key Parameter Registry

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

The columns of the table are:

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

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

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

CBOR type This field contains the CBOR type for the field

description This field contains a brief description for the field

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

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

9.8. Media Type Registration

9.8.1. COSE Security Message

This section registers the "application/cose" and "application/cose+cbor" media types in the "Media Types" registry. [CREF13] These media types are used to indicate that the content is a COSE_MSG.

Type name: application

Subtype name: cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

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

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

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

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

Type name: application

Subtype name: cose+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

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

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

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

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

9.8.2. COSE Key media type

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

Type name: application

Subtype name: cose-key+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

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

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

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

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

Type name: application

Subtype name: cose-key-set+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

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

Interoperability considerations: N/A

Published specification: RFC TBD

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): cbor

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

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: Jim Schaad, ietf@augustcellars.com

Change Controller: IESG

Provisional registration? No

10. Security Considerations

There are security considerations:

1. Protect private keys

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

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

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

5. Considerations about protected vs unprotected header fields.

11. References

11.1. Normative References

[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.

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.", Nov 2007.

[DSS] U.S. National Institute of Standards and Technology, "Digital Signature Standard (DSS)", July 2013.

[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.

[I-D.irtf-cfrg-curves] Langley, A. and R. Salz, "Elliptic Curves for Security", draft-irtf-cfrg-curves-02 (work in progress), March 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.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 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.

[RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, December 2005.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[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.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 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.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

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

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

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

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

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

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", May 2009.

[SP800-56A] Barker, E., Chen, L., Roginsky, A., and M. Smid, "NIST Special Publication 800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", May 2013.

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:

- 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 that 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.

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

- Level 0: Has a content encrypted with AES-GCM using a 128-bit key.
- Level 1: Uses the AES Key wrap algorithm with a 128-bit key.
- Level 3: Uses ECDH Ephemeral-Static direct to generate the level 1 key.

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

```
{
  1: 2,
  2: h'a10101',
  3: {
    -1: h'02d1f7e6f26c43d4868d87ce'
  },
  4: h'64f84d913ba60a76070a9a48f26e97e863e285295a44320878caceb076
3a334806857c67',
  9: [
    {
      3: {
        1: -3
      },
      4: h'5a15dbf5b282ecb31a6074ee3815c252405dd7583e078188',
      9: [
        {
          3: {
            1: "ECDH-ES",
            5: h'6d65726961646f632e6272616e64796275636b406275636b
6c616e642e6578616d706c65',
            4: {
              1: 1,
              -1: 4,
              -2: h'b2add44368ea6d641f9ca9af308b4079aeb519f11e9b8
a55a600b21233e86e68',
              -3: h'1a2cf118b9ee6895c8f415b686d4ca1cef362d4a7630a
31ef6019c0c56d33de0'
            }
          }
        }
      ]
    }
  ]
}
```

Appendix C. Examples

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

Examples may have some features that are in questions but not yet incorporated in the document.

To make it easier to read, the examples are presented using the CBOR's diagnostic notation rather than a binary dump. [CREF14] Using the Ruby based CBOR diagnostic tools at ????, the diagnostic notation

can be converted into binary files using the following command line: (install command is?...)

```
diag2cbor < inputfile > outputfile
```

The examples can be extracted from the XML version of this docuent via an XPath expression as all of the artwork is tagged with the attribute type='CBORdiag'.

C.1. Examples of MAC messages

C.1.1. Shared Secret Direct MAC

This example users the following:

- MAC: AES-CMAC, 256-bit key, trucated to 64 bits
- Key management: direct shared secret
- File name: Mac-04

```
{
  1: 3,
  2: h'a1016f4145532d434d41432d3235362f3634',
  4: h'5468697320697320746865520636f6e74656e742e',
  10: h'd9afa663dd740848',
  9: [
    {
      3: {
        1: -6,
        5: h'6f75722d736563726574'
      }
    }
  ]
}
```

C.1.2. ECDH Direct MAC

This example uses the following:

- MAC: HMAC w/SHA-256, 256-bit key [CREF15]
- Key management: ECDH key agreement, two static keys, HKDF w/ context structure

```
{
  1: 3,
  2: h'a10104',
  4: h'546869732069732074686520636f6e74656e742e',
  10: h'2ba937ca03d76c3dbad30cfcbaeef586f9c0f9ba616ad67e9205d3857
6ad9930',
  9: [
    {
      3: {
        1: "ECDH-SS",
        5: h'6d65726961646f632e6272616e64796275636b406275636b6c61
6e642e6578616d706c65',
        "spk": {
          "kid": "peregrin.took@tuckborough.example"
        },
        "apu": h'4d8553e7e74f3c6a3a9dd3ef286a8195cbf8a23d19558ccf
ec7d34b824f42d92bd06bd2c7f0271f0214e141fb779ae2856abf585a58368b01
7e7f2a9e5ce4db5'
      }
    }
  ]
}
```

C.1.3. Wrapped MAC

This example uses the following:

- MAC: AES-MAC, 128-bit key, truncated to 64 bits
- Key management: AES keywrap w/ a pre-shared 256-bit key

```
{
  1: 3,
  2: h'a1016e4145532d3132382d4d41432d3634',
  4: h'546869732069732074686520636f6e74656e742e',
  10: h'6d1fa77b2dd9146a',
  9: [
    {
      3: {
        1: -5,
        5: h'30313863306165352d346439622d343731622d626664362d6565
66333134626337303337'
      },
      4: h'711ab0dc2fc4585dce27effa6781c8093eba906f227b6eb0'
    }
  ]
}
```

C.1.4. Multi-recipient MAC message

This example uses the following:

- MAC: HMAC w/ SHA-256, 128-bit key
- Key management: Uses three different methods
  1. ECDH Ephemeral-Static, Curve P-521, AES-Key Wrap w/ 128-bit key
  2. RSA-OAEP w/ SHA-256
  3. AES-Key Wrap w/ 256-bit key

```
{
  1: 3,
  2: h'a10104',
  4: h'546869732069732074686520636f6e74656e742e',
  10: h'7aaa6e74546873061f0a7de21ff0c0658d401a68da738dd8937486519
83ce1d0',
  9: [
    {
      3: {
        1: "ECDH-ES+A128KW",
        5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
6d706c65',
        4: {
          1: 1,
          -1: 5,
          -2: h'43b12669acac3fd27898ffba0bcd2e6c366d53bc4db71f909
a759304acfb5e18cdc7ba0b13ff8c7636271a6924b1ac63c02688075b55ef2d61
3574e7dc242f79c3',
          -3: h'812dd694f4ef32b11014d74010a954689c6b6e8785b333d1a
b44f22b9d1091ae8fc8ae40b687e5cfbe7ee6f8b47918a07bb04e9f5b1a51a334
a16bc09777434113'
        }
      },
      4: h'1b120c848c7f2f8943e402cbdbdb58efb281753af4169c70d0126c
0d16436277160821790ef4fe3f'
    },
    {
      3: {
        1: -2,
        5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
6d706c65'
      },
      4: h'46c4f88069b650909a891e84013614cd58a3668f88fa18f3852940
```

```
   a20b35098591d3aacf91c125a2595cda7bee75a490579f0e2f20fd6bc956623bf
   de3029c318f82c426dac3463b261c981ab18b72fe9409412e5c7f2d8f2b5abaf7
   80df6a282db033b3a863fa957408b81741878f466dcc437006ca21407181a016c
   a608ca8208bd3c5a1ddc828531e30b89a67ec6bb97b0c3c3c92036c0cb84aa0f0
   ce8c3e4a215d173bfa668f116ca9f1177505afb7629a9b0b5e096e81d37900e06
   f561a32b6bc993fc6d0cb5d4bb81b74e6ffb0958dac7227c2eb8856303d989f93
   b4a051830706a4c44e8314ec846022eab727e16ada628f12ee7978855550249cc
   b58'
       },
       {
         3: {
           1: -5,
           5: h'30313863306165352d346439622d343731622d626664362d6565
   66333134626337303337'
         },
         4: h'0b2c7cfce04e98276342d6476a7723c090dfdd15f9a518e7736549
   e998370695e6d6a83b4ae507bb'
       }
     ]
   }
```

C.2. Examples of Encrypted Messages

C.2.1. Direct ECDH

This example uses the following:

- CEK: AES-GCM w/ 128-bit key
- Key managment: ECDH Ephemeral-Static, Curve P-256

```
{
  1: 2,
  2: h'a10101',
  3: {
    -1: h'c9cf4df2fe6c632bf7886413'
  },
  4: h'45fce2814311024d3a479e7d3eed063850f3f0b9f3f948677e3ae9869b
cf9ff4e1763812',
  9: [
    {
      3: {
        1: "ECDH-ES",
        5: h'6d65726961646f632e6272616e64796275636b406275636b6c61
6e642e6578616d706c65',
        4: {
          1: 1,
          -1: 4,
          -2: h'98f50a4ff6c05861c8860d13a638ea56c3f5ad7590bbfbf05
4e1c7b4d91d6280',
          -3: h'f01400b089867804b8e9fc96c3932161f1934f4223069170d
924b7e03bf822bb'
        }
      }
    }
  ]
}
```

C.3. Examples of Signed Message

C.3.1. Single Signature

This example uses the following:

- Signature Algorithm: RSA-PSS w/ SHA-384, MGF-1

```
{
  1: 1,
  4: h'546869732069732074686520636f6e74656e742e',
  5: [
    {
      2: h'a2016550533383405581e62696c626f2e62616767696e7340686f
626269746f6e2e6578616d706c65',
      6: h'1b22515f96fd798a331c7b156e90bfea7f558ec6de840e05a8e5f4
b7be44ea1451c48517da7fd216c6143898673c232a96937ebcfb88264a58f5995
82d89cf8a4f20ef35fbfcfd2aad46ad8b99ea6425367afd898de1b712d558b0d2
49d6d180d0b1fb7256140ec3553556f3b5b95a49931a75998dfc23ca905efc7d8
e04deeb92d5936c0824e535aa344396f73913d8a65de0010600270ae5df7f5c8d
52ae525a7642d4c4ff9e219acaa52fd933df003be36b9e3c77ced37129d66745e
2a42baa3d0b3f2675cd51ae8a64fd024d126be5396c91b9236fb5f8548d09881b
b5d40a61c0d342bed9fe8058f36b8722b9e8465dc3b8bfa4f2fd138ce186b73e4
082'
    }
  ]
}
```

C.3.2. Multiple Signers

This example uses the following:

- Signature Algorithm: RSA-PSS w/ SHA-256, MGF-1

- Signature Algorithm: ECDSA w/ SHA-512, Curve P-521

```
{
  1: 1,
  4: h'546869732069732074686520636f6e74656e742e',
  5: [
    {
      2: h'a10129',
      3: {
        5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
6d706c65'
      },
      6: h'028947ac3521f66f2506013007e2cd7b0cb09a209e76ab5b95f751
eb63f5730f1672a282419c49b9653d742577fb6a6cea9ab2e1d4d5d9e786e2240
4760663cc74a1c2c90160af92628e1ebbc3eeba552f757054b691ab17271396b7
ff2d86c100b94a2fce0438c0b50ca70bcdd3074a0f8dc40c2e44e9b26e9093287
b7245ee13171b28ea0f3e291c2cca64aa17f7094aee2be02b5fe5cd2cf343e18c
eec0c763cb76a128df9a9cbfc37b835f6467d98d74505eee1dccc9e6ebf2405ea
1329b41a33eeb13f1bbef3a272e42b3df96cdaea9016663e31ddff4603eb66a88
5c583b53977c1fb9707550717d7387f84616a6670e27d4007b08879109aaf3720
f33'
    },
    {
      3: {
        1: -9,
        5: h'62696c626f2e62616767696e7340686f626269746f6e2e657861
6d706c65'
      },
      6: h'0195345953742c6725352a13cdc55402c895133525c9a3b16bb47d
02ca5f57f8a34aebf47298c602a8feb1dd71d1936886f21029a4142abf38c3aa3
94b3597c2f35c01987c801edc7022c8fddacbf25bc8794b9ffb7cb27f9f346ba4
4db6f5c9b60406530f62b378c5da3e7e2259327f4e55f48271873496497724492
d90ba67a4b65112'
    }
  ]
}
```

Appendix D. COSE Header Algorithm Label Table

This section disappears when we make a decision on password based key management.

name	algorithm	label	CBOR type	description
p2c	PBE	-1	int	
p2s	PBE	-2	bstr	

Appendix E. Document Updates

E.1. Version -00 to -01

- Add note on where the document is being maintained and contributing notes.
- Put in proposal on MTI algorithms.
- Changed to use labels rather than keys when talking about what indexes a map.
- Moved nonce/IV to be a common header item.
- Expand section to discuss the common set of labels used in COSE_Key maps.
- Add a set of straw man proposals for algorithms. It is possible/expected that this text will be moved to a new document.
- Add a set of straw man proposals for key structures. It is possible/expected that this text will be moved to a new document.
- Start marking element 0 in registries as reserved.
- Update examples.

Editorial Comments

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

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

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

[CREF4] JLS: OPEN ISSUE

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

[CREF6] JLS: After looking at this, I am wondering if the type for this should be: [int int]/[int tstr] so that we can keep the major/ minor difference of media-types. This does cost a couple of bytes in the message.

[CREF7] JLS: Need to figure out how we are going to go about creating this registry -or are we going to modify the current mime-content table?

[CREF8] Ilari: I don’t follow/understand this text

[CREF9] JLS: Should this sentence be removed?

[CREF10] JLS: Do we remove this line and just define them ourselves?

[CREF11] JLS: We can really simplify the grammar for COSE_Key to be just the kty (the one required field) and the generic item. The reason to do this is that it makes things simpler. The reason not to do this says that we really need to add a lot more items so that a grammar check can be done that is more tightly enforced.

[CREF12] JLS: Finish the registration process.

[CREF13] JLS: Should we register both or just the cose+cbor one?

[CREF14] JLS: Do we want to keep this as diagnostic notation or should we switch to having "binary" examples instead?

[CREF15] JLS: Need to examine how this is worked out. In this case the length of the key to be used is implicit rather than explicit. This needs to be the case because a direct key could be any length, however it means that when the key is derived, there is currently nothing to state how long the derived key needs to be.

Author’s Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com

ACE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: December 31, 2015

G. Selander
J. Mattsson
F. Palombini
Ericsson
L. Seitz
SICS Swedish ICT

June 29, 2015

Object Security for CoAP (OSCOAP)
draft-selander-ace-object-security-02

Abstract

This memo presents OSCOAP, a scheme for protection of request and response messages of the Constrained Application Protocol (CoAP), using data object security.

Table of Contents

1. Introduction

The Constrained Application Protocol CoAP [RFC7252] was designed with a constrained RESTful environment in mind. CoAP references DTLS [RFC6347] for securing the message exchanges. Two commonly used features of CoAP are store-and-forward and publish-subscribe exchanges, which are problematic to secure with DTLS and transport layer security. As DTLS offers hop-by-hop security, in case of store-and-forward exchanges it necessitates a trusted intermediary. On the other hand, securing publish-subscribe CoAP exchanges with DTLS requires the use of the keep-alive mechanism which incurs additional overhead and actually takes away most of the benefits of asynchronous communication.

The pervasive monitoring debate has illustrated the need to protect data also from trustworthy intermediary nodes as they can be compromised. The community has reacted strongly to the revelations, and new solutions must consider this attack [RFC7258] and include encryption by default.

This memo presents OSCOAP, a data object based communication security solution complementing DTLS and supporting secure messaging end-to-end across intermediary nodes. OSCOAP may be used in very constrained settings where DTLS cannot be supported. OSCOAP can also be combined with DTLS thus enabling, for example, end-to-end security of CoAP payload in combination with hop-by-hop protection of the entire CoAP message during transport between end-point and intermediary node.

1.1 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]. These words may also appear in this document in lowercase, absent their normative meanings.

Certain security-related terms are to be understood in the sense defined in RFC 4949 [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify". For "signature", see below.

RESTful terms, such as "resource" or "representation", are to be understood as used in HTTP [RFC7231] and CoAP.

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

JSON Web Signature (JWS), JOSE Header, JWS Payload, and JWS Signature are defined in [RFC7515]. JSON Web Encryption (JWE), JWE AAD, JWE Ciphertext, and JWE Authentication Tag are defined in [RFC7516].

Secure Message (SM), Secure Signed Message (SSM), and Secure Encrypted Message (SEM) are message formats defined in this memo. The Compact Secure Message (CSM) format is defined in Appendix C. The Sig and Enc options are CoAP options defined in this memo.

Note that "signature" as in "JSON Web Signature" and the derived terms "Secure Signed Message" and "Sig" option may refer either to digital signature using private key of an asymmetric key pair, or Message Authentication Code using a shared key. In other occurrences we use the term as defined in [RFC4949], meaning digital signature.

Excluded Authenticated Data (EAD) is defined in this memo (see Sections 4.1.2). Transaction Identifier (TID) is defined in this memo (see Section 4.1.1).

COSE is defined in [I-D.schaad-cose-msg].

2. Background

The background for this work is provided by the use cases and problem description in [I-D.ietf-ace-usecases] and [I-D.gerdes-ace-actors]. The focus of this memo is on end-to-end security in constrained environments in the presence of intermediary nodes.

For constrained-node networks there may be several reasons for messages to be cached or stored in one node and later forwarded. For example, connectivity between the nodes may be intermittent, or some node may be sleeping at the time when the message should have been forwarded (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.1). Also, the architectural model or protocol applied may require an intermediary node which breaks security on transport layer (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.2). Examples of intermediary nodes include forward proxies, reverse proxies, pub-sub brokers, HTTP-CoAP cross-proxies, and SMS servers.

On a high level, end-to-end security in this setting encompasses:

1. Protection against eavesdropping and manipulation of resource representations in intermediary nodes;

2. Protection against message replay;

3. Protection of authorization information ("access tokens") in transport from an Authorization Server to a Resource Server via

a Client, or other intermediary nodes which could gain from changing the information;

4. Allowing a client to verify that a response comes from a certain server and is the response to a particular request;

5. Protection of the RESTful method used by the client, or the response code used by the server. For example if a malicious proxy replaces the client requested GET with a DELETE this must be detected by the server;

6. Protection against eavesdropping of meta-data of the request or response, including CoAP options such as for example Uri-Path and Uri-Query, which may reveal some information on what is requested.

From the listed examples, there are two main categories of security requirements and corresponding solutions. The first category deals essentially with protecting the CoAP payload (1-3).

The second category deals with protecting an entire CoAP message, targeting also CoAP options and header fields (4-6). The next section formulates security requirements for the two categories, which correspond to two modes of OSCOAP denoted Mode:PAYL and Mode:COAP, respectively.

3. End-to-end Security in Presence of Intermediary Nodes

For high-level security requirements related to resource access, see section 8.7 of [I-D.gerdes-ace-actors]. This section defines the specific requirements that address the two categories of examples identified in the previous section, taking into account potential intermediary nodes.

In the case of CoAP payload only protection (Mode:PAYL), the end-to-end security requirements apply to payload data, such as Resource Representations:

a. The payload shall be integrity protected and should be encrypted end-to-end from sender to receiver.

b. It shall be possible for an intended receiver to detect if it has received this message previously, i.e. replay protection.

Note that a Mode:PAYL message may have multiple recipients. For example, in the case of a proxy that is caching responses used to serve multiple clients, or in a publish-subscribe setting with multiple subscribers to a given publication.

In the case of protecting specific Client-Server CoAP message exchanges (Mode:COAP), potentially passing via intermediary nodes, there are additional end-to-end security requirements:

c. The CoAP options which are not intended to be changed by an intermediary node shall be integrity protected between Client and Server.

d. The CoAP options which are not intended to be read by an intermediary node shall be encrypted between Client and Server.

e. The CoAP header field "Code" shall be integrity protected between Client and Server.

f. A Client shall be able to verify that a message is the response to a particular request the Client made.

The requirements listed above can be met by encryption, integrity protection and replay protection. What differs between the modes is the actual data that is protected, i.e. CoAP payload data only or also other CoAP message data. This memo specifies a common "Secure Message" format that can be used to wrap either payload only (Mode:PAYL) or also additional selected CoAP message fields (Mode:COAP), and be sent as part of the message.

4. Secure Message

There exist already standardized and draft content formats for cryptographically protected data such as CMS [RFC5652], JWS [RFC7515], JWE [RFC7516], and COSE [I-D.schaad-cose-msg].

Current CMS and JWx objects are undesirably large for very constrained devices, and can lead to packet fragmentation in constrained-node networks due to limited frame sizes, and to problems with limited storage capacity on constrained devices due to limited RAM. First estimates with COSE render more compact objects, see Appendix E for a discussion of message format overhead and minimum message expansion. For example, COSE header for a Message Authentication Code object encodes to 37 bytes, while the same header with JWS results in 74 bytes.

Thus, the candidate message format for use in OSCOAP is COSE [I-D.schaad-cose-msg]. Pending a stable version of COSE this draft uses multiple formats and their terminology to illustrate how the message format is applied and processed. It is the intention to replace these with a profile of one single compact secure message format in a future version of this draft.

None of the message formats listed about provide support for replay protection, but it is noted section 10.10 of [RFC7515] that one way to thwart replay attacks is to include a unique transaction identifier and have the recipient verify that the message has not been previously received or acted upon.

We use the term Secure Message (SM) format to refer to a content format for cryptographically protected data which includes a unique transaction identifier, and some other common data as specified in Section 4.1.1.

This memo uses JOSE content formats as a model to specify format and processing of messages. The terms Secure Signed Message (SSM) format and Secure Encrypted Message (SEM) format to refer to Secure Message formats supporting integrity protection only and additional encryption, analogous to JWS and JWE, respectively. Appendix B shows how to profile JOSE objects to become Secure Message formats. Appendix C shows how to profile COSE objects to become Secure Message formats.

4.1 Secure Message format

A Secure Message (SM) SHALL consist of Header, Body and Tag.

4.1.1 Secure Message Header

The following parameters SHALL be included in the SM Header:

- Algorithm. This parameter identifies the cryptographic algorithm(s) used to protect the Secure Message. In case of SSM it has the same semantics as the JOSE Header Parameter "alg" defined in Section 4.1.1 of [RFC7515]. In case of SEM, "direct key agreement" (corresponding to the JWE "alg" = "dir") is assumed, and the encryption algorithm corresponds to the JOSE Header Parameter "enc" (Section 4.1.2 of [RFC7516]). However, the cipher suites are not limited to AEAD algorithms but also include symmetric key encryption combined with private key signature.

- Context Identifier. This parameter identifies the sender and the security context/key(s) used together with the Algorithm to protect the message. For Mode:COAP, the Context Identifier typically identifies the sending party and different resources are identified by their Uri-Path. For Mode:PAYL, the Context Identifier may identify the resource itself. The structure of this identifier is unspecified.

- Sequence Number. The Sequence Number parameter enumerates the Secure Messages protected using the security context identified by the Context Identifier, and is used for replay protection and uniqueness of nonce. The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The ordered sequence (Sequence Number, Context Identifier) is called Transaction Identifier (TID), and SHALL be unique for each SM.

4.1.2 Secure Message Body

Analogously to JWS and JWE, the SM Body contains what is being protected. The SM Body is different for SSM and SEM.

In order to obtain a compact representation, certain data is integrity protected but excluded from the Secure Message. Such data is referred to as Excluded Authenticated Data (EAD). To further reduce message size, the unencrypted part of the SM Body may be "detached" from the Secure Message, see sections 4.1.2.1 and 4.1.2.2.

The assumption behind excluding integrity protected data from the SM, or detaching integrity protected but not encrypted parts of the SM during transport, is that the data in question is known to the receiver, e.g. because it is established beforehand or because it is transported as part of the CoAP message carrying the Secure Message.

4.1.2.1 Secure Signed Message Body

For SSM, the Body consists of the payload data which is integrity protected, analogously to the JWS Payload. Detached Content is defined to mean that the Body is removed from the Secure Message, analogously to Appendix F of [RFC7515]. Hence a SSM with Detached Content consists of Header and Tag.

4.1.2.2 Secure Encrypted Message Body

Analogously to JWE, the terms Plaintext, Ciphertext and Additional Authenticated Data (AAD) are used for the SEM. The Body of a SEM consists of Ciphertext, the encrypted Plaintext as defined by the Algorithm, and Additional Authenticated Data (AAD) which is integrity protected by the Algorithm as defined by the Cipher Suite. For SEM Detached Content is defined to mean that the AAD is removed from the Secure Message. Hence a SEM with Detached Content consists of the Header, Ciphertext and Tag.

4.1.3 Secure Message Tag

The SM Tag consists of the Signature / Message Authentication Code value as defined by the Algorithm, calculated over the SM Header, SM Body and EAD (if present). The content of EAD depends on the Mode, see 5.1.3 and 5.2

5. Message Protection

This section describes what is protected in a Secure Message and how it depends on the defined Modes "COAP" and "PAYL". The use of Mode:COAP is signaled with the presence of the options Sig or Enc defined in this section. The differences in SM Body and SM Tag as a function of Mode are described below.

Both formats SSM and SEM defined in the previous section are applicable to both Modes. For any Secure Message Mode, the SEM format SHALL be used by default. Examples of SSM and SEM are given in Appendix F.

5.1 CoAP Message Protection (Mode:COAP)

Referring to examples 4-6 in Section 2 and requirements a-f in Section 3, this section presents how to protect individual CoAP messages including options and header fields, as well as request-response message exchanges, using the Secure Message format. This is called Mode:COAP. An endpoint receiving a CoAP request containing a Secure Message with Mode:COAP MUST respond with a CoAP message containing a Secure Message with Mode:COAP.

Since slightly different message formats are used for integrity protection only (SSM), and additional encryption (SEM), these cases are treated separately.

5.1.1 The Sig Option

In order to integrity protect CoAP message exchanges including options and headers, a new CoAP option is introduced: the Sig option, containing a SSM Mode:COAP object. Endpoints supporting this scheme MUST check for the presence of a Sig option, and verify the SSM as described in Section 5.1.1.2 before accepting a message as valid.

5.1.1.1 Option Structure

The Sig option indicates that certain CoAP Header Fields, Options, and Payload (if present) are integrity and replay protected using a Secure Signed Message (SSM). The Sig option SHALL contain a SSM with Detached Content (see Section 4.1.2.1).

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 1 illustrates the structure of this option.

No.	C	U	N	R	Name	Format	Length *)
TBD	x		x		Sig	opaque	12-TBD

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 1: The Sig Option

*) Length is essentially Length(SSM Header) + Length(SSM Tag). The minimum length is estimated in Appendix E. The maximum length depends on actual message format selected and is TBD.

5.1.1.2 Integrity Protection and Verification

A CoAP endpoint composing a message with the Sig option SHALL process the SSM and produce the SSM Tag, as defined in 5.1.1.3 and 5.1.3, analogously to the specification for producing a JWS object as described in Section 5.1 of [RFC7515] (cf. Appendix B). In addition, the sending endpoint SHALL process the Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Sig option SHALL first recreate the SSM Body as described in Section 5.1.1.3, and then verify the SSM Tag as described in Section 5.1.3, analogously to the specification for verifying a JWS object as described in Section 5.2 of [RFC7515] (cf. Appendix B). In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

5.1.1.3 SSM Body

The SSM Body of SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;
- all CoAP options present which are marked as IP in Table 3 (Appendix A), in the order as given by the option number (each

Option with Option Header including delta to previous IP-marked Option which is present); and

o the CoAP Payload (if any).

5.1.2 The Enc Option

In order to encrypt and integrity protect CoAP messages, a new CoAP option is introduced: the Enc option, indicating the presence of a SEM Mode:COAP object in the CoAP message, containing the encrypted part of the CoAP message. Endpoints supporting this scheme MUST check for the presence of an Enc option, and verify the SEM as described in 5.1.2.2 before accepting a message as valid.

5.1.2.1 Option Structure

The Enc option indicates that certain CoAP Options and Payload (if present) are encrypted, integrity and replay protected using a Secure Encrypted Message (SEM) with Detached Content (see Section 4.1.2.2). The structure of a CoAP message with an Enc option is described in Section 5.1.2.4.

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 2 illustrates the structure of this option.

No.	C	U	N	R	Name	Format	Length *)
TBD	x		x		Enc	opaque	0 or 12-TBD

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 2: The Enc Option

*) Length indicates in this case the additional length added to the total length of all CoAP options. If the CoAP message has Payload, then the Enc option is empty, otherwise it contains the SEM (see Section 5.1.2.4). In the latter case, the SEM Ciphertext contains the encrypted CoAP Options (see Section 5.1.2.3), which are thus excluded from plaintext part of the message. Hence the additional length is essentially Length(SEM Header) + Length(SEM Tag). The minimum length is estimated in Appendix E. The maximum length

depends on actual message format selected and is TBD.

5.1.2.2 Encryption and Decryption

A CoAP endpoint composing a message with the Enc option SHALL process the SEM and produce the SEM Ciphertext and SEM Tag, as defined in 5.1.2.3 and 5.1.3, analogously to the specification for producing a JWE object as described in Section 5.1 of [RFC7516] (cf. Appendix B). In addition, the sending endpoint SHALL process the Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Enc option SHALL first recreate the SEM Body as described in Section 5.1.2.3, and then decrypt and verify the SEM analogously to the specification for verifying a JWE object as describe in Section 5.2 of [RFC7516] (cf. Appendix B). In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

5.1.2.3 SEM Body

The SEM Plaintext SHALL consist of the following data, formatted as a CoAP message without Header consisting of:

- all CoAP Options present which are marked as E in Table 3 (see Appendix A), in the order as given by the Option number (each Option with Option Header including delta to previous E-marked Option); and
- the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

The SEM Additional Authenticated Data SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;
- all CoAP options present which are marked as IP and not marked as E in Table 2 (see Appendix A), in the order as given by the Option number (each Option with Option Header including delta to previous such Option).

5.1.2.4 CoAP Message with Enc Option

An unprotected CoAP message is encrypted and integrity protected by

means of an Enc option and a SEM. The structure and format of the protected CoAP message being sent instead of the unprotected CoAP message is now described.

The protected CoAP message is formatted as an ordinary CoAP message, with the following Header, Options and Payload:

- The CoAP header SHALL be the same as the unprotected CoAP message.
- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message, and the Enc option. The options shall be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).
- If the unprotected CoAP message has no Payload then the Enc option SHALL contain the SEM with Detached Content. If the unprotected CoAP message has Payload, then the SEM option SHALL be empty and the Payload of the CoAP message SHALL be the SEM with Detached Content. The Payload is prefixed by the one-byte Payload Marker (0xFF).

5.1.3 SM Tag

This section describes the SM Tag for Mode:COAP, which applies both to SEM and SSM. The SM Tag is defined in 4.1.3. If the message is a CoAP Request, then EAD SHALL be empty. If the message is a CoAP Response, then EAD SHALL consist of the TID of the associated CoAP Request.

5.2 Payload Only Protection (Mode:PAYL)

Referring to examples 1-3 in Section 2 and requirements a and b in Section 3, the case of only protecting CoAP payload using the Secure Message format is now discussed. This is called Mode:PAYL.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the relevant SM format (SSM or SEM) Mode:PAYL. The SSM (SEM) SHALL be protected (encrypted) and verified (decrypted) as described in 5.1.1.2 (5.1.2.2), including replay protection as described in section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

For Mode:PAYL, the EAD SHALL be empty. Hence, the SM Tag is calculated over the SM Header and SM Body.

A CoAP message where the Payload is wrapped as a Mode:PAYL object is indicated by setting the option Content-Format to application/smpayl. A CoAP client may request a response containing such a payload wrapping by setting the option Accept to application/smpayl. (See Section 8.)

5.3 Replay Protection and Freshness

In order to protect from replay of messages and verify freshness of responses, a CoAP endpoint supporting OSCOAP SHALL maintain Transaction Identifiers (TIDs) of sent and received Secure Messages (see section 4.1.1).

5.3.1 Replay Protection

An endpoint SHALL maintain a TID and associated security context/key(s) for each other endpoint it receives messages from, and one TID and associated security context/key(s) for protecting sent messages. Depending on use case, an endpoint MAY maintain a sliding receive window for Sequence Numbers associated to TIDs in received messages, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message a sending endpoint SHALL step the Sequence Number of the associated send TID and SHALL include it in the SM Header parameter Sequence Number as defined in section 4.1.1. However, if the Sequence Number counter wraps, the client must first acquire a new TID and associated security context/key(s). The latter is out of scope of this memo.

A receiving endpoint SHALL verify that the Sequence Number received in the SM Header is greater than the Sequence Number in the TID for received messages (or within the sliding window and not previously received) and update the TID (window) accordingly.

5.3.2 Freshness

If a CoAP server receives a valid Secure Message request in Mode:COAP, then the response SHALL include the TID of the request as EAD, as defined in section 5.1.3. If the CoAP client receives a Secure Message response in Mode:COAP, then the client SHALL verify the signature by reconstructing SM Body and using the TID of its own associated request as EAD, as defined in section 5.1.3.

6. Security Considerations

In scenarios with proxies, gateways, or caching, DTLS only protects data hop-by-hop meaning that these intermediary nodes can read and modify information. The trust model where all participating nodes are considered trustworthy is problematic not only from a privacy perspective but also from a security perspective as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture weak.

DTLS protects the entire CoAP message including Header, Options and Payload, whereas Mode:COAP protects the message fields described in Appendix A. The cost for DTLS providing this protection is the overhead in e.g. additional messages, processing, memory incurred by the DTLS Handshake protocol, which can be omitted in use cases where key establishment can be provided by other means.

Mode:COAP provides point to point encryption, integrity and replay protection, and freshness of response. Payload as well as relevant options and header field Code are protected.

Mode:PAYL only protects payload and only gives replay protection (not freshness), but allows additional use cases such as point to multi-point interactions including publish-subscribe, reverse proxies and proxy caching of responses. In case of symmetric keys the receiver does not get data origin authentication, which requires a digital signature using a private asymmetric key. Mode:PAYL SHALL NOT be used in cases where the CoAP header field Code needs to be integrity protected.

Blockwise transfers in CoAP [I-D.ietf-core-coap-block] can be applied both to Mode:COAP and Mode:PAYL. With Mode:COAP each block and the Block options are integrity protected. Hence each individual block can be securely verified by the receiver, retransmission securely requested etc. With Mode:PAYL the entire payload is encapsulated in a Secure Message which is partitioned into blocks which are sent with unprotected CoAP. The receiver is able to verify the integrity of the payload but only after the last block containing the signature/MAC is received, and if the verification fails the entire message needs to be resent. However, if the verification succeeds, then the transmission in Mode:PAYL has less computational and packet overhead since only one signature/MAC was generated and sent. As CoAP blockwise transfer with Mode:PAYL is prone to Denial of Service attacks, it should only be used for exchanges where this threat can be mitigated, for example within a local area network where link-layer security is activated.

The Version header field is not integrity protected to allow backwards compatibility with future versions of CoAP. Considering this, it may in theory be possible to launch a cross-version attack, e.g. something analogous to a bidding down attack. Future updates of CoAP would need to take this into account.

The use of sequence numbers for replay protection introduces the problem related to wrapping of the counter. The alternatives also have issues: very constrained devices may not be able to support accurate time or generate and store large numbers of random nonces. The requirement to change key at counter wrap is a complication, but it also forces the user of this specification to think about implementing key renewal.

Independently of message format, and whether the target is CoAP message protection or payload only protection, this specification needs to be complemented with a procedure whereby the client and the server establish the keys used for wrapping and unwrapping the Secure Message. One way to address key establishment is to assume that there is a trusted third party which can support client and server, such as the Authorization Server in [I-D.gerdes-ace-actors]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server which can be used in the Secure Message exchange. Similarly, the Authorization Server may, on behalf of the server, notify the client of server supported ciphers, in order to facilitate the usage of OSCOAP in deployments with multiple supported cryptographic algorithms.

The security contexts required for SSM and SEM are different. For a SSM, the security context is essentially Algorithm, Context Identifier, Sequence Number and Key. For a SEM it is also required to have a unique Initialization Vector for each message. The Initialization Vector SHALL be the concatenation of a Salt (4 bytes unsigned integer) and the Sequence Number. The Salt SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it in each message. For example, the Salt may be established by the same means as keys are established. For a SEM, the security context is essentially Algorithm, Context Identifier, Salt, Sequence Number and Key.

7. Privacy Considerations

End-to-end integrity protection provides certain privacy properties, e.g. protection of communication with sensor and actuator from manipulation which may affect the personal sphere. End-to-end encryption of payload and certain CoAP options provides additional

protection as to the content and nature of the message exchange.

The headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and response (Token). Plaintext options could also reveal information, e.g. lifetime of measurement (Max-age), or that this message contains one data point in a sequence (Observe).

8. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[this document]" with the RFC number of this specification.

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Sig	[[this document]]
TBD	Enc	[[this document]]

This document registers the following value in the CoAP Content Format registry established by [RFC7252].

Media Type: application/smpayl

Encoding: -

Id: 70

Reference: [this document]

9. Acknowledgements

Klaus Hartke has independently been working on the same problem and a similar solution: establishing end-to-end security across proxies by adding a CoAP option. We are grateful to Malisa Vucinic for providing helpful and timely comments.

10. References

10.1 Normative References

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

[RFC6347] Rescorla, E., and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

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

[RFC7258] Farrell, S., and H. Tschofenig, "Pervasive Monitoring Is an Attack", RFC 7258, May 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.

10.2 Informative References

[I-D.gerdes-ace-actors]
Gerdes, S., Seitz, L., G. Selander, Bormann, C. "An Arhitecture for Authorization in Constrained Environments", draft-gerdes-ace-actors-05 (work in progress), April 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-04 (work in progress), March 2015.

[I-D.schaad-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-schaad-cose-msg-00 (work in progress), June 2015.

[I-D.ietf-core-coap-block]
Bormann, C., and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-17 (work in progress), July 2014.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

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

RFC 5652, September 2009.

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

[RFC7231] Fielding, R., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

Appendix A. Which CoAP Header Fields and Options to Protect

In the case of CoAP Message Protection (Mode:COAP) as much as possible of the CoAP message is protected. However, not all CoAP header fields or options can be encrypted and integrity protected, because some are intended to be read or changed by an intermediary node.

A.1 CoAP Header Fields

The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length may be changed by a proxy and thus SHALL neither be integrity protected nor encrypted. Example 5 in Section 2 shows that the Code SHALL be integrity protected. The Version parameter SHALL neither be integrity protected nor encrypted (see Section 6).

A.2 CoAP Options

This section describes what options need to be integrity protected and encrypted. On a high level, all CoAP options must be encrypted by default, unless intended to be read by an intermediate node; and integrity protected, unless intended to be changed by an intermediate node.

However, some special considerations are necessary because CoAP defines certain legitimate proxy operations, because the security information itself may be transported as an option, and because different processing is performed for SSM and SEM.

A.2.1 Integrity Protection

CoAP options which are not intended to be changed by an intermediate node MUST be integrity protected:

- CoAP options which are Safe-to-Forward SHALL be integrity protected, the only exception being the security options Enc and Sig. See Table 3.

- Block1, Block2 are Unsafe but not intended to be modified by

intermediaries and hence SHALL be integrity protected.

CoAP options which are intended to be modified by a proxy can be divided into two categories, those that are intended to change in a predictable way, and those which are not. The following options are of the latter kind and SHALL NOT be integrity protected:

- Max-Age, Observe: These options may be modified by a proxy in a way that is not predictable for client and server.

The remaining options may be modified by a proxy, but when they are, the change is predictable. Therefore it is possible to define "invariants" which can be integrity protected.

A.2.1.1 Proxy-Scheme

A Forward Proxy is intended to replace the URI scheme with the content of the Proxy-Scheme option. The Proxy-Scheme option is defined to be an invariant with respect to the following processing:

- If there is a Proxy-Scheme present, then the client MUST integrity protect the Proxy-Scheme option.
- If there is no Proxy-Scheme option present the client SHALL integrity protect the Proxy-Scheme option set to the URI scheme used in the message sent.
- The server SHALL insert the Proxy-Scheme option with the name of the URI scheme the message was received with before verifying the integrity.

A.2.1.2 Uri-*

For options related to URI of resource (Uri-Host, Uri-Port, Uri-Path, Uri-Query, Proxy-Uri) a Forward Proxy is intended to replace the Uri-* options with the content of the Proxy-Uri option.

The Proxy-Uri option is defined to be an invariant with respect to the following processing (applying to a SSM, for SEM see next section):

- If there is a Proxy-Uri present, then the client MUST integrity protect the Proxy-Uri option and the Uri-* options MUST NOT be integrity protected.
- If there is no Proxy-Uri option present, then the client SHALL compose the full URI from Uri-* options according to the method

described in section 6.5 of [RFC7252]. The SM Tag is calculated on the following message, modified compared to what is sent:

- All Uri-* options removed
- A Proxy-Uri option with the full URI included

- The server SHALL compose the URI from the Uri-* options according to the method described in section 6.5 of [RFC7252]. The so obtained URI is placed into a Proxy-Uri option, which is included in the integrity verification.

A.2.2 Encryption

All CoAP options MUST be encrypted, except the options below which MUST NOT be encrypted:

- Max-Age, Observe: This information is intended to be read by a proxy.
- Enc, Sig: These are the security-providing options.
- Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.

In the case of a SEM, the Proxy-Uri MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not intended to be revealed to a Forward Proxy.

A.2.3 Summary

Table 3 summarizes which options are encrypted and integrity protected, if present.

In a SSM, options marked with "a" and "b" are composed into a URI as described above and included as the Proxy-Uri option which is part of the SSM Body. In a SEM, options marked "a" are composed into a URI as described above and included as the Proxy-Uri option in the SEM Additional Authenticated Data.

No.	C	U	N	R	Name	Format	Length	E	IP
1	x			x	If-Match	opaque	0-8	x	x

3	x	x	-		Uri-Host	string	1-255		a
4				x	ETag	opaque	1-8	x	x
5	x				If-None-Match	empty	0	x	x
6		x	-		Observe	uint	0-3		
7	x	x	-		Uri-Port	uint	0-2		a
8				x	Location-Path	string	0-255	x	x
11	x	x	-	x	Uri-Path	string	0-255	x	b
12					Content-Format	uint	0-2	x	x
14		x	-		Max-Age	uint	0-4		
15	x	x	-	x	Uri-Query	string	0-255	x	b
17	x				Accept	uint	0-2	x	x
20				x	Location-Query	string	0-255	x	x
23	x	x	-		Block2	uint	0-3	x	x
27	x	x	-		Block1	uint	0-3	x	x
28			x		Size2	uint	0-4	x	x
35	x	x	-		Proxy-Uri	string	1-1034		x
39	x	x	-		Proxy-Scheme	string	1-255		x
60			x		Size1	uint	0-4	x	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt, IP=Integrity Protect.

Table 3: Protected CoAP options in Mode=COAP.

Appendix B. JOSE Profile of SM

This section defines profiles of JWS and JWE complying with the Secure Message format (see Section 4.1). The use of compact serialization is assumed.

B.1 JWS as Secure Signed Message

The JOSE Header of JWS contains the mandatory parameter "alg", defined in Section 4.1.1 of [RFC7515], which corresponds to the parameter Algorithm of the Secure Message.

A JWS is a Secure Message if the JOSE Header includes

- the new parameter "cid" defined in B.4, and
- the new parameter "seq" defined in B.3.

An SSM with Detached Content corresponds to a JWS with JOSE Header and JWS Signature; i.e. no JWS Payload.

B.2 JWE as Secure Encrypted Message

In case of JWE, the SM Header parameters of a JWE consists of the JOSE Header Parameters and JWE Initialization Vector (IV).

The JOSE Header of JWE contains the mandatory parameter "enc", defined in Section 4.1.2 of [RFC7516], which corresponds to the parameter Algorithm of the Secure Message. The JOSE Header also contains the mandatory parameter "alg", the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). It is also assumed that plaintext compression (zip) is not used.

A JWE is a Secure Message if the JOSE Header includes

- o the new parameter "cid" defined in B.4, and
- o the IV contains the Sequence Number and a Salt (see Section 6).

An SEM with Detached Content corresponds to a JWE with JOSE Header, JWE Initialization Vector, JWE Ciphertext and JWE Authentication Tag; i.e. no JWE AAD.

B.3 "seq" (Sequence Number) Header Parameter

The Sequence Number, corresponding to the Secure Message parameter with the same name (Section 4.1.1), SHALL be an integer represented as a byte string. Only the significant bytes are sent (initial bytes with zeros are removed). The start sequence number SHALL be 0. For a given key, "seq" MUST NOT be used more than once.

The parameter "seq" SHALL be marked as critical using the "crit" header parameter (see section 4.1.11 of [RFC7515]), meaning that if a receiver does not understand this parameter it must reject the message.

B.4 "cid" (Context Identifier) Header Parameter

The Context Identifier, corresponding to the Secure Message parameter with the same name (Section 4.1.1), SHALL be a unique byte string identifying the security context of the sending party. The parameter "cid" SHALL be marked as critical.

Appendix C. Compact Secure Message

For constrained environments it is important that the message expansion due to security overhead is kept at a minimum. As an attempt to assess what this minimum expansion could be, this section defines an optimized bespoke Secure Message format (Section 4.1) called the Compact Secure Message (CSM) format. This is intended as a benchmark for COSE [I-D.schaad-cose-msg].

The Compact Secure Message (CSM) format is depicted in Figure 4.

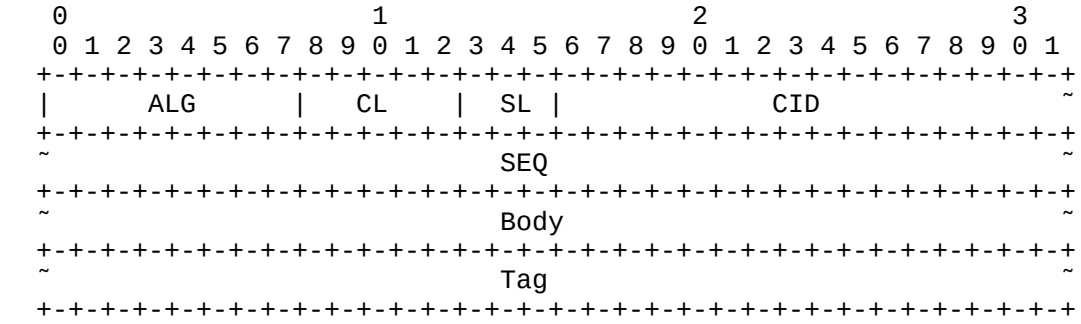


Figure 4: Compact Secure Message format

The CSM Header (see Section 4.1.1.) consists of 2 bytes of fixed length parameters and two variable length parameters, Context Identifier (CID) and Sequence Number (SEQ). The Header parameters are (compare Table 5):

- Algorithm (ALG). This parameter consists of an encoding of the ciphersuite used in the Secure Message. The encoding is TBD.
- CID Length (CL). This parameter consist of a length indication of the header parameter Context Identifier. The actual length of CID is CL + 1 bytes.
- SEQ Length (SL). This parameter consist of a length indication of the header parameter Sequence Number. The actual length of SEQ is SL + 1 bytes.
- Context Identifier (CID). This parameter identifies the security context/key(s) used to protect the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).
- Sequence Number (SEQ). This parameter consists of the sequence number used by the sender of the Secure Message. Only the

significant bytes are sent (initial bytes with zeros are removed).

Name	Parameter	Length
ALG	Algorithm	8 bits
CL	Context Identifier Length	5 bits
SL	Sequence Number Length	3 bits
CID	Context Identifier	CL + 1: 1-32 bytes
SEQ	Sequence Number	SL + 1: 1-8 bytes

Table 5: CSM Header Parameters.
The minimum CSM Header is 4 bytes.

The TID consists of the concatenation of SEQ and CID, in that order, formatted as in the CSM format (initial bytes with zeros are removed).

The content of CSM Body depends on whether it is a SSM or a SEM (see Section 4.1.2) which is determined by the Algorithm. This version of the draft focuses on Secure Message with Detached Content. Hence, the SSM Body is empty and the SEM Body consists of the Ciphertext. In the former case, the length of the CSM Body is 0. In the latter case, the length of the CSM Body equals the sum of the lengths of the present CoAP options marked encrypted in Table 3 and the length of the payload of the unprotected CoAP message.

The CSM Tag contains the MAC/Signature as determined from the Algorithm. The length is determined by ALG.

Appendix D. COSE Profile of SM

This section defines a profile of the 00-version of COSE [I-D.schaad-cose-msg] complying with the Secure Message format (see Section 4.1) and supporting the two modes of operation Mode:COAP and Mode:PAYL. In the last subsection we elaborate on possible optimizations.

D.1 COSE_Sign or COSE_mac as Secure Signed Message

SSM corresponds to COSE_MSG msg_type 1 (COSE_Sign) or 3 (COSE_mac).

D.1.1 COSE_Sign as Secure Signed Message

A COSE_MSG of type COSE_Sign is a Secure Message if its fields are defined as follows (see example in Appendix E.2).

The "Headers" field of COSE_Sign MUST contain the field "protected" and this field MUST include the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1).

The mandatory "signatures" array contains one "COSE_signature" item which contains a "protected" field and the mandatory "signature" field. The "protected" field includes:

- the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);
- the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The mandatory "signature" field contains the computed signature value.

A SSM with digital signature and Detached Content corresponds to COSE_sign with "Headers" and "signatures" fields; i.e. no "payload" field.

D.1.2 COSE_mac as Secure Signed Message

A COSE_MSG of type COSE_mac is a Secure Message if its fields are defined as follows (see example in Appendix E.1).

The "Headers" field of COSE_mac object MUST contain the "protected" field, the "recipient" field and the mandatory "tag" field. The "protected" field MUST include:

- the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);
- the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1).

The "recipients" array contains one "COSE_encrypt_a" item (section 5 of [I-D.schaad-cose-msg]), which contains an "unprotected" field that includes:

- o the "alg" parameter corresponding to the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). (Appendix A of [I-D.schaad-cose-msg]);

- o the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The mandatory "tag" field contains the MAC value.

A SSM with MAC and Detached Content corresponds to a COSE_sign with "Headers", "recipients" and "tag" fields; i.e. no "payload" field.

D.2 COSE_encrypt as Secure Encrypted Message

SEM with AEAD algorithm corresponds to COSE_MSG msg_type 2 (COSE_encrypt). A COSE_MSG of type COSE_encrypt [I-D.schaad-cose-msg] is a Secure Message if its fields are defined as follows (see example in Appendix E.3).

The "Headers" field of COSE_encrypt MUST contain the "protected" field, the "recipient" field, the "cipherText" field and depending on the algorithm used, the "iv" field.

The "iv" corresponds to the Initialization Vector, which contains a Salt (see Section 6) and Sequence Number as defined in section 4.1.1. For some algorithms, it is mandatory to include the "iv" field and hence the Salt is sent in each message.

The "protected" field includes:

- o the "alg" parameter which corresponds to the parameter Algorithm of the Secure Message (see section 4.1.1);

- o the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see section 4.1.1). This parameter is present only if the "iv" field is not present in the COSE_encrypt structure.

The "recipients" array contains one "COSE_encrypt_a" item as defined in section 5 of [I-D.schaad-cose-msg], which contains an "unprotected" field that includes:

- o the "alg" parameter corresponding to the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). (Appendix A of [I-D.schaad-cose-msg]);

- o the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see section 4.1.1);

The "cipherText" field contains the encrypted plain text, as defined in section 5 of [I-D.schaad-cose-msg].

A SEM with Detached Content corresponds to a COSE_encrypt with "Headers", "recipients", optionally "iv" and "cipherText" fields; i.e. no "aad" field.

D.3 COSE optimizations

This section lists potential optimizations of COSE [I-D.schaad-cose-msg] for the purpose of reducing message size and improving performance in constrained node networks. The message sizes resulting from the first two optimizations are presented in Appendix E (as "modified COSE").

1. For COSE_encrypt and COSE_mac, there is a ’recipient’ field (see section 6 of [I-D.schaad-cose-msg]). This field is intended for a setting when the sender is aware of the recipients of the message, and can wrap keys for these recipients. This is not necessarily true in the use cases targeting constrained devices and thus one possible optimization is to remove the ’recipient’ field. The Context Identifier "cid" can be carried in the Header, preferably in the protected field, to avoid both protected and unprotected fields causing additional overhead. (An alternative is to define "tid", Transaction Identifier, as an array consisting of "seq" and "cid".)

2. Analogous to other key values, one-byte keys/labels can be assigned to the new parameters defined in this document and cipher suites adapted to constrained device processing. For example: "cid" = 11, "seq" = 12, and "AES-CCM" = 14.

3. The combination of secret key encryption and digital signature is well founded in the use cases. A solution based on wrapping one COSE message into another creates substantial overhead (see difference between modified COSE and CSM in Table 11 of Appendix E.4). A valuable optimization would be to define combined cipher suites and security contexts, and corresponding "alg" and "cid" parameters. An example would be 128-bit AES and particular curve parameters for a 64 bytes ECDSA signature.

4. Digitally signed messages have the largest absolute overhead due to the size of the signature (see Appendices E.2 and E.4).

Whereas certain MACs can be securely truncated, signatures cannot. Signature schemes with message recovery allow some remedy since they allow part of the message to be recovered from the signature itself and thus need not be sent. The effective size of the signature could in this way be considerably reduced, which would have a large impact on the message size (compare size of signature and total overhead in Tables 9 and 11). A valuable optimization is thus to support signature schemes with message recovery.

Appendix E. Comparison of message sizes

This section gives some examples of overhead incurred with JOSE, with the current proposal for COSE at the time of writing [I-D.schaad-cose-msg], and with CSM. CSM should be viewed as a lower bound for COSE. Message sizes are also listed for a modified version of COSE implementing some of the optimizations described in Appendix D.3.

Motivated by the use cases, there are four different kinds of protected messages that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The latter is relevant e.g. for proxy-caching and publish-subscribe with untrusted intermediary (see Appendix F.2). The sizes estimated for selected algorithms are detailed in the subsections.

The size of the header is shown separately from the size of the MAC/signature, since JWS/JWE has no provisions for truncating it. Compact serialization for both JWS and JWE is assumed. For CSM the encoding of algorithms is assumed as in COSE. An 8-byte Context Identifier and a 3-byte Sequence Number are used throughout all examples. To make it easier to read, COSE objects are represented using CBOR’s diagnostic notation rather than a binary dump.

E.1 SSM: Message Authentication Code

This example is based on HMAC-SHA256, with truncation to 16 bytes. For JWS the following header is used:

```
{"alg":"HS256","cid":0xa1534e3c5fdc09bd,"seq":0x112233}
```

which encodes to a size of 74 bytes in Base64url, and the 32 bytes of HS256 MAC encode to 43 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The same object in COSE gives:

{1:3, 2:{1:4, "seq":h'112233'}, 9:[{3:{"cid":h'a1534e3c5fdc09bd', 1:-6}}], 10:MAC}, where MAC is the truncated 16-byte MAC.

The COSE object encodes to a total size of 53 bytes.

In a modified version of COSE, with no 'recipient' field (see section 6 of [I-D.schaad-cose-msg])and protected "cid" in the header, 1-byte key values are assigned to "cid" and "seq", for exemple: "cid" = 11 and "seq" = 12. The equivalent COSE object would be:

{1:3, 2:{1:4, 12:h'112233', 11:h'a1534e3c5fdc09bd'}, 10:MAC}, where MAC is the truncated 16-byte MAC.

This modified COSE object encodes to a total size of 40 bytes.

For CSM the same header is represented by 13 bytes.

Table 6 summarizes these results.

Scheme	Header	MAC	Total Overhead
JWS	74 B	43 B	119 bytes
COSE	37 B	16 B	53 bytes
mod-COSE	24 B	16 B	40 bytes
CSM	13 B	16 B	29 bytes

Table 6: Comparison of JWS, COSE, modified COSE and CSM for HMAC-SHA256.

E.2 SSM: Digital Signature

This example is based on ECDSA, with a signature of 64 bytes.

For JWS the following header is used:

{"alg":"ECDSA","cid":0xa1534e3c5fdc09bd,"seq":0x112233}

which encodes to a size of 74 bytes in Base64url, and the 64 bytes of signature encode to 86 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The same object in COSE gives:

{1:1, 2:{"seq":h'112233'}, 5:[{2:{1:-7,"cid":h'a1534e3c5fdc09bd'}, 6:SIG}]}, where SIG is the 64-byte signature.

The COSE object encodes to a total size of 100 bytes.

In a modified version of COSE, 1-byte key values are assigned to "cid" and "seq", for exemple: "cid" = 11 and "seq" = 12. The equivalent COSE object would be:

{1:1, 2:{12:h'112233'}, 5:[{2:{1:-7,11:h'a1534e3c5fdc09bd'}, 6:SIG}, where SIG is the 64-byte signature.

The COSE object encodes to a total size of 94 bytes.

For CSM the same header is represented by 13 bytes.

Table 7 summarizes these results.

Scheme	Header	Tag	Total Overhead
JWS	74 B	86 B	162 bytes
COSE	36 B	64 B	100 bytes
mod-COSE	30 B	64 B	94 bytes
CSM	13 B	64 B	77 bytes

Table 7: Comparison of JWS, COSE, modified COSE and CSM for 64 byte ECDSA signature.

E.3 SEM: Authenticated Encryption

This example is based on both AES-128-CCM-8 and AES-128-GCM. Since the former is not supported by JOSE, we use the latter for comparison between JOSE and COSE.

For JWE it is assumed that the IV is generated from the Sequence Number and some previously agreed upon Salt. This means it is not required to explicitly send the whole IV in the CSM format, but also that the JWE and COSE formats may omit the Sequence Number.

The JWE header

{"alg":"dir","cid":0xa1534e3c5fdc09bd,"enc":"A128GCM"}

encodes to a size of 72 bytes in Base64url, while the necessary 12 byte IV for GCM mode is expanded to 16 bytes by encoding. The 16 bytes of the authentication tag expand to 22 bytes. The concatenation marks add 3 bytes to the total overhead.

The corresponding COSE object is:

{1:2, 2:{1:1}, 7:IV, 4:TAG, 9:[{3:{"cid":h'a1534e3c5fdc09bd', 1:-6}}]}, where IV is the 12-byte IV and TAG the 16-byte authentication tag

The COSE object encodes to a total size of 59 bytes.

Table 8 summarizes these results.

Scheme	Header	IV	Tag	Total Overhead
JWE	72 B	16 B	22 B	113 bytes
COSE	31 B	12 B	16 B	59 bytes

Table 8: Comparison of JWE and COSE for AES-GCM.

The same calculation have been done using CCM mode instead of GCM, and adding a 3-byte "seq". The COSE object is represented as follows:

{1:2, 2:{1:"AES-CCM","seq":h'112233'}, 4:TAG, 9:[{3:{"cid":h'a1534e3c5fdc09bd', 1:-6}}]}, where TAG is the 16-byte authentication tag.

The COSE object encodes to a total size of 52 bytes.

In a modified version of COSE, the 'recipient' field is removed (see section 6 of [I-D.schaad-cose-msg])and "cid" is protected in the header. 1-byte key values are assigned to "cid", "seq" and "AES-CCM", for exemple: "cid" = 11, "seq" = 12 and "AES-CCM" = 14. The equivalent COSE object would be:

{1:2, 2:{1:14,12:h'112233'}, 4:TAG, 9:[{3:{11:h'a1534e3c5fdc09bd', 1:-6}}]}, where TAG is the truncated 8-byte authentication tag.

This modified COSE object encodes to a total size of 39 bytes.

For CSM, the corresponding header for AES-128-CCM-8, including the 8 byte Sequence Number, is represented by 13 bytes and the tag is truncated to 8 Bytes.

Table 9 summarizes these results.

Scheme	Header	Tag	Total Overhead
COSE	44 B	16 B	60 bytes
mod-COSE	31 B	8 B	39 bytes
CSM	13 B	8 B	21 bytes

Table 9: Comparison of COSE, modified COSE and CSM for AES-CCM.

E.4 SEM: Symmetric Encryption + Digital Signature

This example is based on AES-128 and ECDSA with 64 bytes signature. JOSE and COSE require this to be a nested encapsulation of one object into another, here illustrated with a digitally signed AEAD protected object.

For JWS the following header is used:

{"alg":"ECDSA","cid":0xa1534e3c5fdc09bd,"seq":0x112233}

which encodes to a size of 74 bytes in Base64url, and the 64 bytes of signature encode to 86 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The payload of the JWS object is a JWE object with the following header:

{"alg":"dir","cid":0xa1534e3c5fdc09bd,"enc":"A128GCM"}

which encodes to a size of 72 bytes in Base64url, while the necessary 12 byte IV for GCM mode is expanded to 16 bytes by encoding. The 16 bytes of the authentication tag expand to 22 bytes. The concatenation marks add 3 bytes to the total overhead.

The total size of the JWS object is 275 bytes.

The same object in COSE gives:

{1:1, 2:{"seq":h'112233'}, 4:{1:2, 2:{1:1}, 7:IV, 4:TAG, 9:[{3:{"cid":h'a1534e3c5fdc09bd', 1:-6}}]}, 5:[{2:{1:-7,"cid":h'a1534e3c5fdc09bd'}, 6:SIG}]}, where SIG is the 64-byte signature, IV is the 12-byte IV and TAG the 16-byte authentication tag.

The COSE object encodes to a total size of 160 bytes.

Table 10 summarizes these results.

Scheme	Header	Sig	Payload	Total Overhead
JWS	74 B	86 B	113 B	275 bytes
COSE	37 B	64 B	59 B	160 bytes

Table 10: Comparison of JWS and COSE for nested AES-GCM within ECDSA.

The same calculation have been done using CCM mode instead of GCM, and adding a 3-byte "seq" in the protected header.

The COSE object is represented as follows:

{1:1, 2:{"seq":h'112233'}, 4:{1:2, 2:{1:"AES-CCM","seq":h'112233'}, 4:TAG, 9:[{3:{"cid":h'a1534e3c5fdc09bd', 1:-6}}]}, 5:[{2:{1:-7,"cid":h'a1534e3c5fdc09bd'}, 6:SIG}]}, where SIG is the 64-byte signature and TAG is the 16-byte authentication tag.

The COSE object encodes to a total size of 153 bytes.

In a modified version of COSE, the 'recipient' field is removed (see section 6 of [I-D.schaad-cose-msg])and "cid" is protected in the header. 1-byte key values are assigned to "cid", "seq" and "AES-CCM", for exemple: "cid" = 11, "seq" = 12 and "AES-CCM" = 14. The equivalent COSE object would be:

{1:1, 2:{12:h'112233'}, 4:{1:2, 2:{1:14,12:h'112233'}, 4:TAG, 9:[{3:{11:h'a1534e3c5fdc09bd', 1:-6}}]}, 5:[{2:{1:-7,11:h'a1534e3c5fdc09bd'}, 6:SIG}]}, where SIG is the 64-byte signature and TAG is the 8-byte truncated authentication tag.

This modified COSE object encodes to a total size of 134 bytes.

For CSM we assume that an (AES, ECDSA) cipher suite has been defined, and that the "cid" identifies the context used by both the algorithms. Then the corresponding header is represented by 13 bytes, and the signature by 64 bytes.

Table 11 summarizes these results.

Scheme	Header	Sig	Payload	Total Overhead
COSE	36 B	64 B	52 B	153 bytes
mod-COSE	30 B	64 B	39 B	134 bytes
CSM	13 B	64 B	0 B	77 bytes

Table 11: Comparison of nested AES-CCM within ECDSA (COSE, modified COSE) and combined AES-ECDSA (CSM).

Appendix F. Examples

This section gives examples of how to use the new options and message formats defined in this memo.

F.1 CoAP Message Protection

This section illustrates Mode:COAP. The message exchange assumes there is a security context established between client and server. One key is used for each direction of the message transfer. The intermediate node detects that the CoAP message contains a SM Mode:COAP object (Sig or Enc option is set) and thus forwards the message as it cannot serve a cached response.

F.1.1 Integrity Protection of CoAP Message Exchange

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Sig option. The example illustrates a client closing a lock and getting a confirmation that the lock is closed. Code, Uri-Path and Payload of the request and Code of the response are integrity protected (and other message fields, see Appendix A).

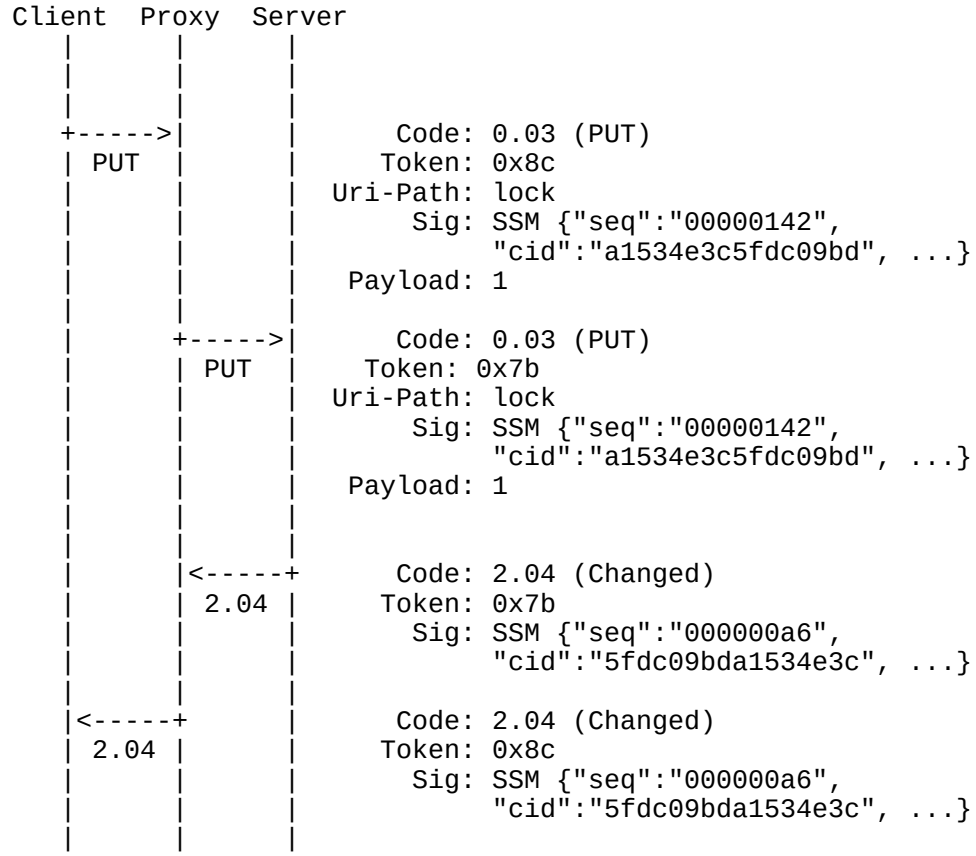


Figure 8: CoAP PUT protected with Sig/SSM (Mode:COAP)

The Context Identifier is an identifier indicating which security context was used to integrity protect the message, and may be used as an identifier for a secret key or a public key. (It may e.g. be the hash of a public key.)

The server and client can verify that the Sequence Number has not been received and used with this key before, and since Mode is COAP, the client can additionally verify the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 5.3).

The SSM also contains the Tag as specified in the Algorithm (not shown in the Figure).

This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the Sig option. If there is no compelling

reason why the CoAP message should be in plaintext, then the Enc option MUST be used.

F.1.2 Additional Encryption of CoAP Message

Here is an example of a GET request/response message exchange passing an intermediate node protected with the Enc option. The example illustrates a client requesting a blood sugar measurement resource (GET /glucose) and receiving the value 220 mg/dl. Uri-Path and Payload are encrypted and integrity protected. Code is integrity protected only (see Appendix A).

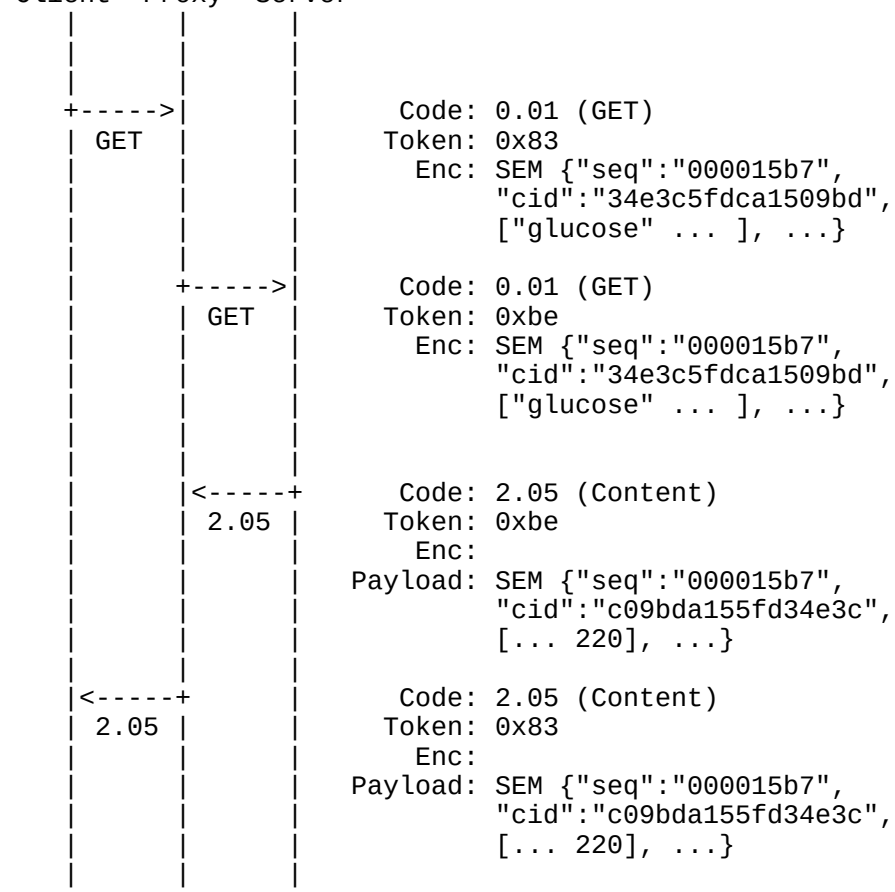


Figure 9: CoAP GET protected with Enc/SEM (Mode:COAP). The bracket [...] indicates encrypted data.

Since the request message (GET) does not support payload, the SEM is

carried in the Enc option. Since the response message (Content) supports payload, the Enc option is empty and the SEM is carried in the payload.

The Context Identifier is a hint to the receiver indicating which security context was used to encrypt and integrity protect the message, and may be used as an identifier for the AEAD secret key. One key is used for each direction of the message transfer.

The server and client can verify that the Sequence Number has not been received and used with this key before, and since Mode:COAP the client can additionally verify the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 5.3).

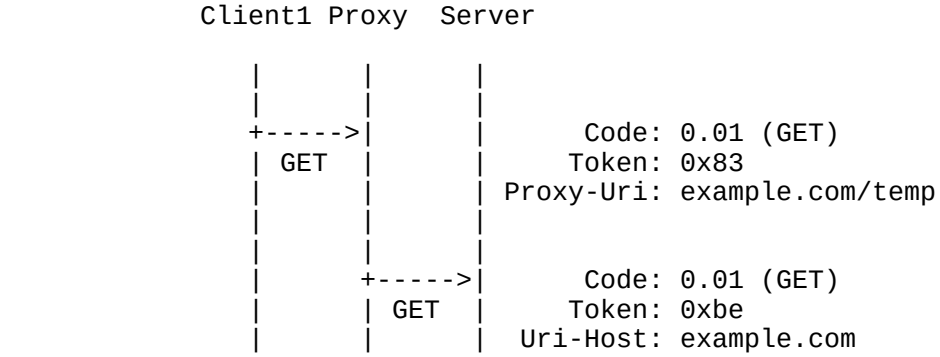
The SEM also contains the Tag as specified by the Algorithm (not shown in the Figure).

F.2 Payload Protection

This section gives examples that illustrate Mode:PAYL. This mode assumes that only the intended receiver(s) has the relevant security context related to the resource. In case of a closed group of recipients of the same object, e.g. in Information-Centric Networking or firmware update distribution, it may be necessary to support symmetric key encryption in combination with digital signature.

F.2.1 Proxy Caching

This examples applies e.g. to closed user groups of a single data source. The example outlines how a proxy forwarding request and response of one client can cache a response whose payload is a SEM object, and serve this response to another client request, such that both clients can verify integrity and non-replay.



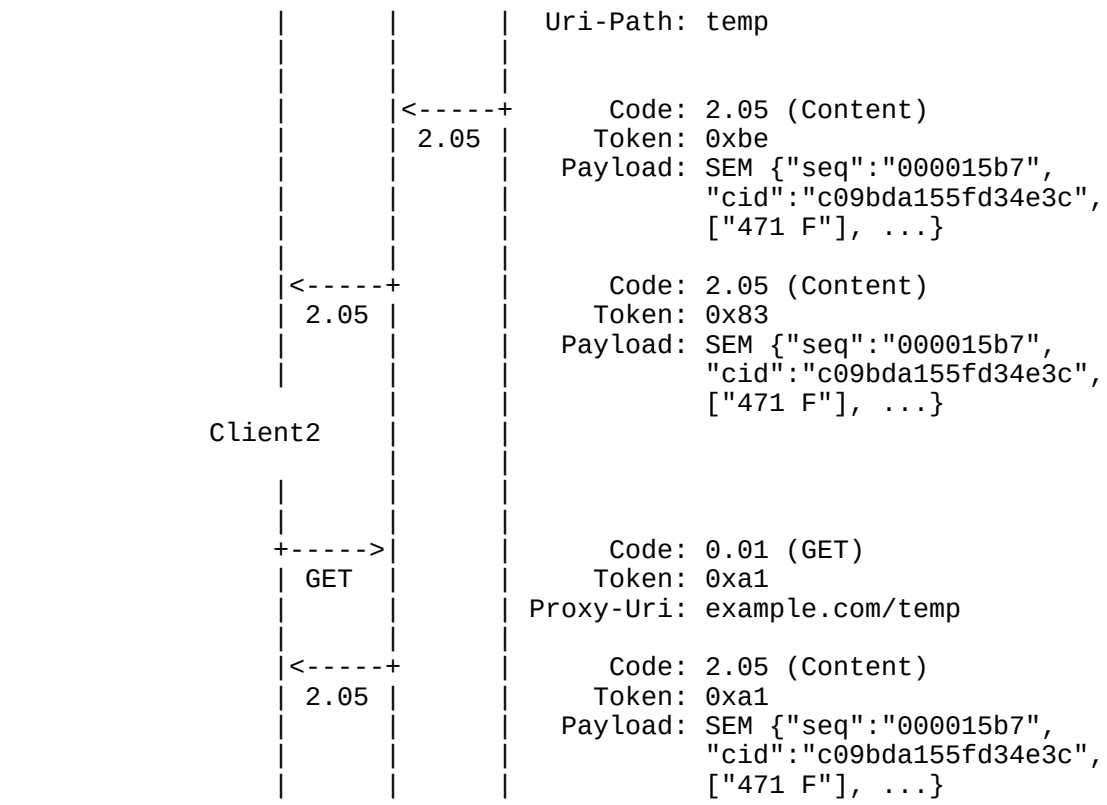


Figure 10: Proxy caching protected with SEM (Mode:PAYL)

F.2.2 Publish-Subscribe

This example outlines a publish-subscribe setting where the payload is integrity and replay protected end-to-end between Publisher and Subscriber. The example illustrates a subscription registration and a new publication of birch pollen count of 300 per cubic meters. The PubSub Broker can define the Observe count arbitrarily (as could any intermediary node, even in Mode:COAP), but cannot manipulate the Sequence Number without being noticed.

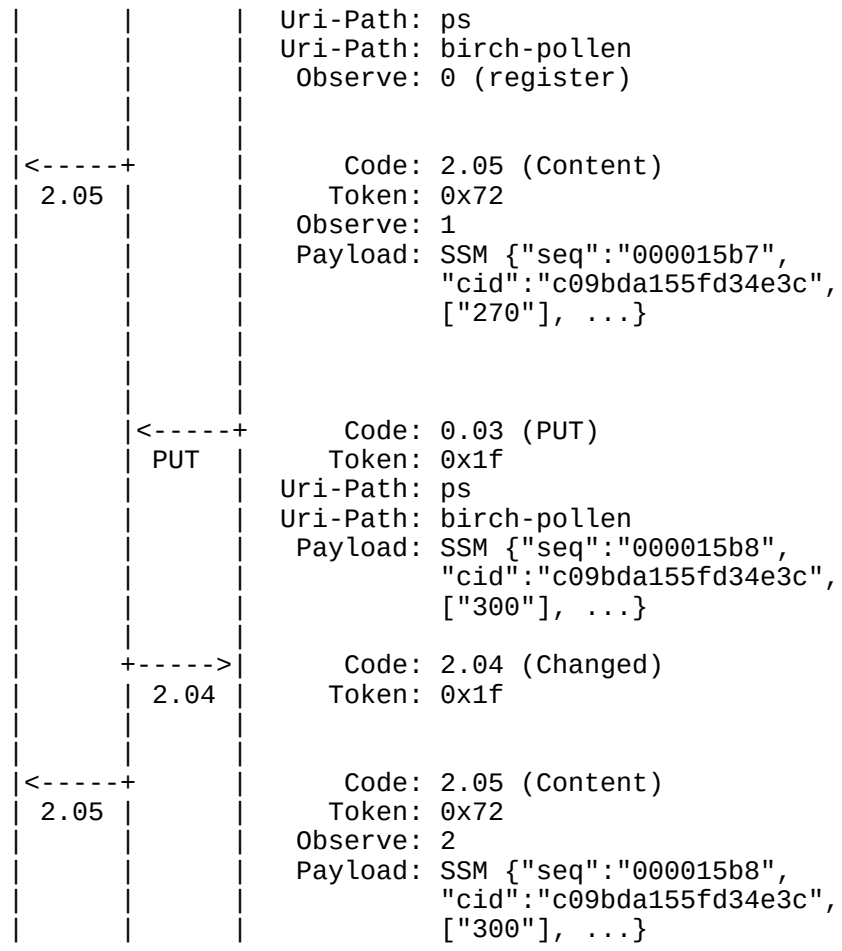


Figure 11: Publish-subscribe protected with SSM (Mode:PAYL)

This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the SSM in Mode:PAYL. If there is no compelling reason why the payload should be in plaintext, then SEM MUST be used.

F.2.3 Transporting Authorization Information

This example outlines the transportation of authorization information from a node producing (Authorization Server, AS) to a node consuming (Resource Server, RS) such information. Authorization information may for example be an authorization decision with respect to a Client (C) accessing a Resource to be enforced by RS. See [I-D.seitz-ace-core-authz] and Section 8.4-8.6 of [I-D.gerdes-ace-actors].

Here, C is clearly not trusted with modifying the information, but may need to be involved in mediating the authorization information to the RS, for example, because AS and RS does not have direct connectivity. So end-to-end security is required and object security ("access tokens") is the natural candidate.

This example considers the authorization information to be encapsulated in a SEM Mode:PAYL object, generated by AS. How C accesses the SEM is out of scope for this example, it may e.g. be using CoAP. C then requests RS to configure the authorization information in the SEM by doing POST to /authorize. This particular resource has a default access policy that only new messages signed by AS are authorized. RS thus verifies the integrity and sequence number by using the existing security context for the AS, and responds accordingly, a) or b), see Figure 12.

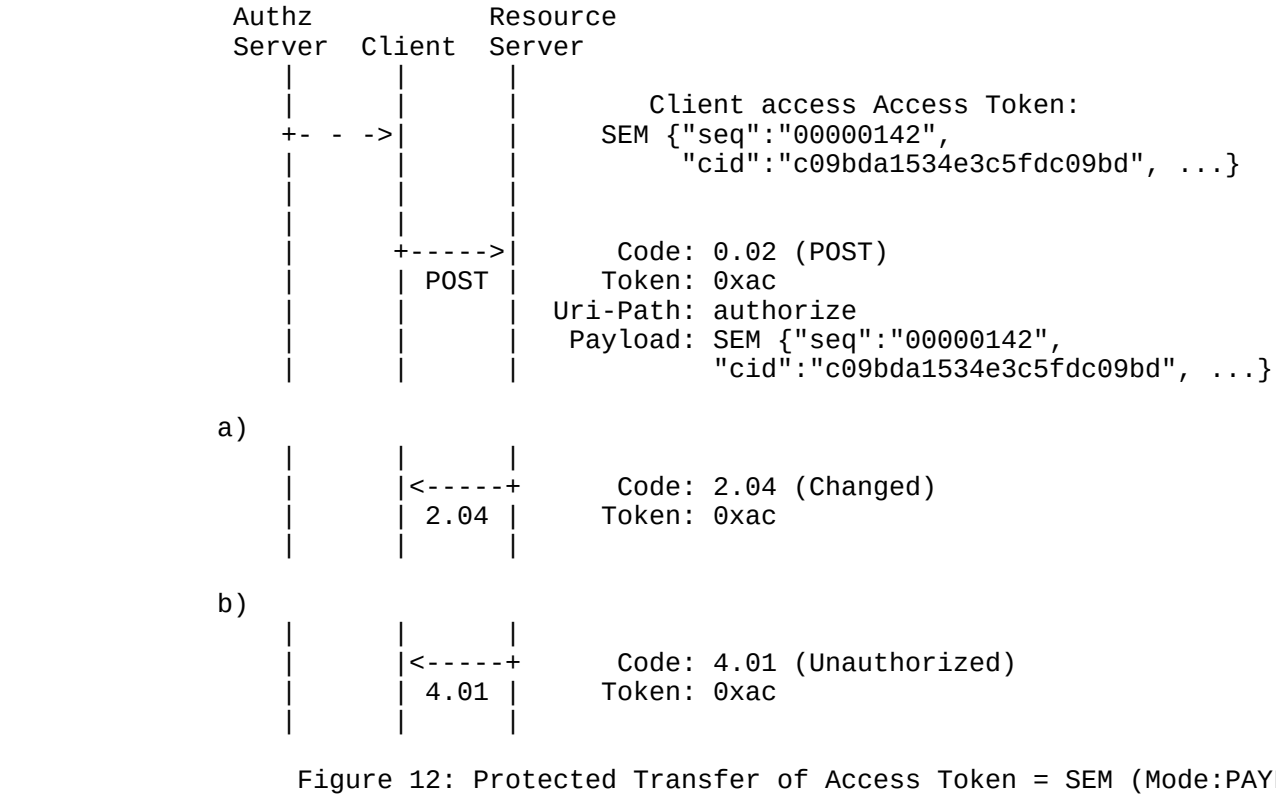


Figure 12: Protected Transfer of Access Token = SEM (Mode:PAYL)

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com

John Mattsson
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: john.mattsson@ericsson.com

Francesca Palombini
Ericsson
Farogatan 6
16480 Kista
SWEDEN
EMail: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se