CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 22, 2017

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

December 19, 2016

Object Security of CoAP (OSCOAP)
draft-ietf-core-object-security-01

Abstract

This memo defines Object Security of CoAP (OSCOAP), a method for application layer protection of message exchanges with the Constrained Application Protocol (CoAP), using the CBOR Object Signing and Encryption (COSE) format. OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options, and header fields, as well as a secure binding between CoAP request and response messages. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo.

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 June 22, 2017.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web application protocol, designed for constrained nodes and networks [RFC7228]. CoAP specifies the use of proxies for scalability and efficiency. At the same time CoAP references DTLS [RFC6347] for security. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of the CoAP payload and metadata, in transit between client and server. The proxy can also inject, delete, or reorder packages without being protected or detected by DTLS.

This memo defines Object Security of CoAP (OSCOAP), a data object based security protocol, protecting CoAP message exchanges end-to-end, across intermediary nodes. An analysis of end-to-end security for CoAP messages through intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs], this specification addresses the forwarding case.

The solution provides an in-layer security protocol for CoAP which does not depend on underlying layers and is therefore favorable for providing security for "CoAP over foo", e.g. CoAP messages passing over both unreliable and reliable transport [I-D.ietf-core-coap-tcp-tls], CoAP over IEEE 802.15.4 IE [I-D.bormann-6lo-coap-802-15-ie].

OSCOAP builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg], providing end-to-end encryption, integrity, and replay protection. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo. The solution transforms an unprotected CoAP message into a protected CoAP message in the following way: the unprotected CoAP message is protected by including payload (if present), certain options, and header fields in

a COSE object. The message fields that have been encrypted are removed from the message whereas the Object-Security option and the COSE object are added. We call the result the "protected" CoAP message. Thus OSCOAP is a security protocol based on the exchange of protected CoAP messages (see Figure 1).

Figure 1: Sketch of OSCOAP

OSCOAP provides protection of CoAP payload, certain options, and header fields, as well as a secure binding between CoAP request and response messages. OSCOAP provides replay protection, but like DTLS, OSCOAP only provides relative freshness in the sense that the sequence numbers allows a recipient to determine the relative order of messages. For applications having stronger demands on freshness (e.g. control of actuators), OSCOAP needs to be augmented with mechanisms providing absolute freshness [I-D.mattsson-core-coap-actuators].

OSCOAP may be used in extremely constrained settings, where DTLS cannot be supported. Alternatively, OSCOAP can be combined with DTLS, thereby enabling 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. Examples of the use of OSCOAP are given in Appendix B.

The message protection provided by OSCOAP can alternatively be applied only to the payload of individual messages. We call this object security of content (OSCON) and it is defined in Appendix C.

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.

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641]. Readers are also expected to be familiar with [RFC7049] and understand [I-D.greevenbosch-appsawg-cbor-cddl]. Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

2. The Object-Security Option

The Object-Security option indicates that OSCOAP is used to protect the CoAP message exchange. The protection is achieved by means of a COSE object included in the protected CoAP message, as detailed in Section 5.

The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable. Figure 2 illustrates the structure of the Object-Security option.

A CoAP proxy SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original client's request. The Object-Security option is included in the cache key for backward compatibility with proxies not recognizing the Object-Security option. The effect of this is that messages with the Object-Security option will never generate cache hits. To further prevent caching, a Max-Age option with value zero SHOULD be added to the protected CoAP responses.

No.	C	U	N	R	Name	Format	Length
TBD	x				Object-Security	opaque	0-

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

Figure 2: The Object-Security Option

The length of the Object-Security option depends on whether the unprotected message allows payload, on the set of options that are included in the unprotected message, the length of the integrity tag, and the length of the information identifying the security context.

- If the unprotected message allows payload, then the COSE object is the payload of the protected message (see Section 6.2 and Section 6.4), and the Object-Security option has length zero. An endpoint receiving a CoAP message with payload, that also contains

a non-empty Object-Security option SHALL treat it as malformed and reject it.

- o If the unprotected message does not allow payload, then the COSE object is the value of the Object-Security option and the length of the Object-Security option is equal to the size of the COSE object. An endpoint receiving a CoAP message without payload, that also contains an empty Object-Security option SHALL treat it as malformed and reject it.

Note that according to [RFC7252], new Methods and Response Codes should specify if the payload is optional, required or not allowed (Section 12.1.2) in the message, and in case this is not defined the sender must not include a payload (Section 5.5). Thus, in this case, the COSE object MUST be the value of the Object-Security option.

More details about the message overhead caused by the Object-Security option are given in Appendix A.

3. The Security Context

OSCOAP uses COSE with an Authenticated Encryption with Additional Data (AEAD) algorithm. The specification requires that client and server establish a security context to apply to the COSE objects protecting the CoAP messages. In this section we define the security context, and also specify how to derive the initial security contexts in client and server based on common shared secret and a key derivation function (KDF).

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. Each security context is identified by a Context Identifier. A Context Identifier that is no longer in use can be reassigned to a new security context.

For each endpoint, the security context is composed by a "Common Context", a "Sender Context" and a "Recipient Context". The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Each endpoint has a unique ID used to derive its Sender Context, this identifier is called "Sender ID". The Recipient Context is derived with the other endpoint's ID, which is called "Recipient ID". The Recipient ID is thus the ID of the endpoint from which a CoAP message is received. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus the two security contexts identified by the same

Context Identifiers in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 3."

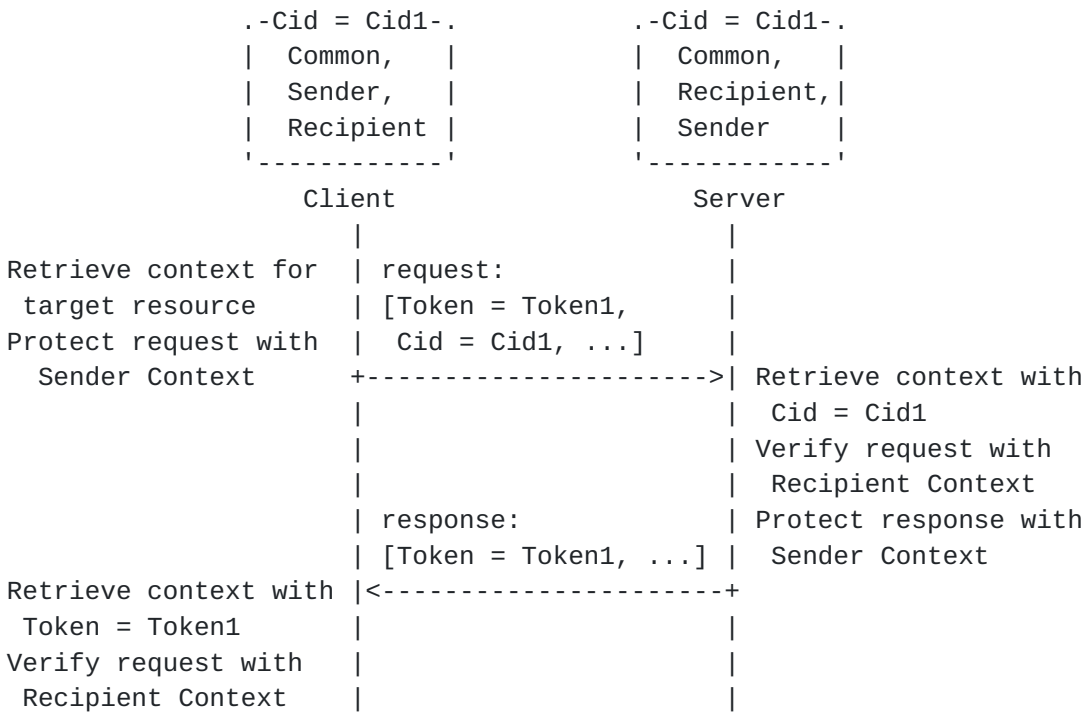


Figure 3: Retrieval and use of the Security Context

The Common Context contains the following parameters:

- Context Identifier (Cid). Variable length byte string that identifies the security context. Its value is immutable once the security context is established.

- Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Its value is immutable once the security context is established.

- Base Key (master_secret). Variable length, uniformly random byte string containing the key used to derive traffic keys and IVs. Its value is immutable once the security context is established.

The Sender Context contains the following parameters:

- Sender ID. Variable length byte string identifying the endpoint itself. Its value is immutable once the security context is established.

- Sender Key. Byte string containing the symmetric key to protect messages to send. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Sender IV. Byte string containing the fixed context IV [I-D.ietf-cose-msg]) to protect messages to send. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Sender Sequence Number. Non-negative integer enumerating the COSE objects that the endpoint sends using the context. Used as partial IV [I-D.ietf-cose-msg] to generate unique nonces for the AEAD. Maximum value is determined by Algorithm.

The Recipient Context contains the following parameters:

- Recipient ID. Variable length byte string identifying the endpoint messages are received from. Its value is immutable once the security context is established.

- Recipient Key. Byte string containing the symmetric key to verify messages received. Length is determined by the Algorithm. Its value is immutable once the security context is established.

- Recipient IV. Byte string containing the context IV to verify messages received. Length is determined by Algorithm. Its value is immutable once the security context is established.

- Recipient Replay Window. The replay protection window for messages received.

The 3-tuple (Cid, Sender ID, Partial IV) is called Transaction Identifier (Tid), and SHALL be unique for each Base Key. The Tid is used as a unique challenge in the COSE object of the protected CoAP request. The Tid is part of the Additional Authenticated Data (AAD, see Section 5) of the protected CoAP response message, which is how responses are bound to requests.

3.2. Derivation of Security Context Parameters

This section describes how to derive the initial parameters in the security context, given a small set of input parameters. We also give indications on how applications should select the input parameters.

The following input parameters SHALL be pre-established:

- Context Identifier (Cid)

- Base Key (master_secret)
- AEAD Algorithm (Alg)
  - Default is AES-CCM-64-64-128 (value 12)

The following input parameters MAY be pre-established:

- Sender ID
  - Defaults are 0x00 for the endpoint intially being client, and 0x01 for the endpoint initially being server
- Recipient ID
  - Defaults are 0x01 for the endpoint intially being client, and 0x00 for the endpoint initially being server
- Key Derivation Function (KDF)
  - Default is HKDF SHA-256
- Replay Window Size
  - Default is 64

The endpoints MAY interchange the CoAP client and server roles while maintaining the same security context. When this happens, the former server still protects the message to send using the Sender Context, and verifies the message received using its Recipient Context. The same is also true for the former client. The endpoints MUST NOT change the Sender/Recipient ID. In other words, changing the roles does not change the set of keys to be used.

The input parameters are included unchanged in the security context. From the input parameters, the following parameters are derived:

- Sender Key, Sender IV, Sender Sequence Number
- Recipient Key, Recipient IV, Recipient Sequence Number

The EDHOC protocol [I-D.selander-ace-cose-ecdhe] enables the establishment of input parameters with the property of forward secrecy, and negotiation of KDF and AEAD, it thus provides all necessary pre-requisite steps for using OSCOAP as defined here.

3.2.1. Derivation of Sender Key/IV, Recipient Key/IV

Given the input parameters, the client and server can derive all the other parameters in the security context. The derivation procedure described here MUST NOT be executed more than once using the same master_secret and Cid. The same master_secret SHOULD NOT be used with more than one Cid.

The KDF MUST be one of the HKDF [RFC5869] algorithms defined in COSE. The KDF HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key/IV, Recipient Key/IV SHALL be derived using HKDF, and consists of the composition of the HKDF-Extract and HKDF-Expand steps ({{RFC5869}}):

```
output parameter = HKDF(master_secret, salt, info, output_length),
```

where:

- master_secret is defined above
- salt is a string of zeros of the length of the hash function output in octets
- info is a serialized CBOR array consisting of:

```
info = [
    cid : bstr,
    id : bstr,
    alg : int,
    out_type : tstr,
    out_len : uint
]
```

  - id is the Sender ID or Recipient ID
  - out_type is "Key" or "IV"
  - out_len is the key/IV size of the AEAD algorithm

- output_length is the size of the AEAD key/IV in bytes encoded as an 8-bit unsigned integer

For example, if the algorithm AES-CCM-64-64-128 (see Section 10.2 in [I-D.ietf-cose-msg]) is used, output_length for the keys is 128 bits and output_length for the IVs is 56 bits.

3.2.2. Context Identifier

As mentioned, Cid is pre-established. How this is done is application specific, but it is RECOMMENDED that the application uses 64-bits long pseudo-random Cids, in order to have globally unique Context Identifiers. Cid SHOULD be unique in the sets of all security contexts used by all the endpoints. If it is not the case, it is the role of the application to specify how to handle collisions.

If the application has total control of both clients and servers, shorter unique Cids MAY be used. Note that Cids of different lengths can be used by different clients and that e.g. a Cid with the value 0x00 is different from the Cid with the value 0x0000.

In the same phase during which the Cid is established in the endpoint, the application informs the endpoint what resources can be accessed using the corresponding security contexts. Resources that are accessed with OSCOAP are called "protected" resources. The set of resources that can be accessed using a certain security context is decided by the application (resource, host, etc.). The client SHALL save the association resource-Cid, in order to be able to retrieve the correct security context to access a protected resource. The server SHALL save the association resource-Cid, in order to determine whether a particular resource may be accessed using a certain Cid.

3.2.3. Sender ID and Recipient ID

The Sender ID and Recipient ID SHALL be unique in the set of all endpoints using the same security context. Collisions may lead to the loss of both confidentiality and integrity. If random IDs are used, they MUST be long enough so that the probability of collisions is negligible.

3.2.4. Sequence Numbers and Replay Window

The Sender Sequence Number is initialized to 0. The Recipient Replay Window is initiated as described in Section 4.1.2.6 of [RFC6347].

4. Protected CoAP Message Fields

OSCOAP transforms an unprotected CoAP message into a protected CoAP message, and vice versa. This section defines how the unprotected CoAP message fields are protected. OSCOAP protects as much of the unprotected CoAP message as possible, while still allowing forward proxy operations [I-D.hartke-core-e2e-security-reqs].

This section also outlines how the message fields are processed and transferred, a detailed description is provided in Section 6. Message fields of the unprotected CoAP message are either transferred in the header/options part of the protected CoAP message, or in the plaintext of the COSE object. Depending on which, the location of the message field in the protected CoAP message is called "outer" or "inner":

- Inner message field = message field included in the plaintext of the COSE object of the protected CoAP message (see Section 5.1)

- Outer message field = message field included in the header or options part of the protected CoAP message

The inner message fields are encrypted and integrity protected by the COSE object. The outer message fields are sent in plain text but may be integrity protected by including the message field values in the AAD of the COSE object (see Section 5.2).

Note that, even though the message formats are slightly different, OSCOAP complies with CoAP over unreliable transport [RFC7252] as well as CoAP over reliable transport [I-D.ietf-core-coap-tcp-tls].

4.1. CoAP Payload

The CoAP Payload SHALL be encrypted and integrity protected, and thus is an inner message field.

The sending endpoint writes the payload of the unprotected CoAP message into the plaintext of the COSE object (see Section 6.2 and Section 6.4).

The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the unprotected CoAP message (see Section 6.3 and Section 6.5).

4.2. CoAP Header

Many CoAP header fields are required to be read and changed during a normal message exchange or when traversing a proxy and thus cannot be protected between the endpoints, e.g. CoAP message layer fields such as Message ID.

The CoAP header field Code MUST be sent in plaintext to support RESTful processing, but MUST be integrity protected to prevent an intermediary from changing, e.g. from GET to DELETE. The CoAP version number SHALL be integrity protected to prevent potential future version-based attacks. Note that while the version number is

not sent in each CoAP message over reliable transport [I-D.ietf-core-coap-tcp-tls], its value is known to client and server.

Other CoAP header fields SHALL neither be integrity protected nor encrypted. The CoAP header fields are thus outer message fields.

The sending endpoint SHALL copy the header fields from the unprotected CoAP message to the protected CoAP message. The receiving endpoint SHALL copy the header fields from the protected CoAP message to the unprotected CoAP message. Both sender and receiver inserts the CoAP version number and header field Code in the AAD of the COSE object (see section Section 5.2).

4.3. CoAP Options

As with the message fields described in the previous sections, CoAP options may be encrypted and integrity protected, integrity protected only, or neither encrypted nor integrity protected.

Most options are encrypted and integrity protected (see Figure 4), and thus inner message fields. But to allow certain proxy operations, some options have outer values and require special processing. Indeed, certain options may or must have both an inner value and a potentially different outer value, where the inner value is intended for the destination endpoint and the outer value is intended for the proxy.

No.	C	U	N	R	Name	Format	Length	E	A
1	x			x	If-Match	opaque	0-8	x	
3	x	x	-		Uri-Host	string	1-255		x
4				x	ETag	opaque	1-8	x	
5	x				If-None-Match	empty	0	x	
6		x	-		Observe	uint	0-3	*	
7	x	x	-		Uri-Port	uint	0-2		x
8				x	Location-Path	string	0-255	x	
11	x	x	-	x	Uri-Path	string	0-255	x	
12					Content-Format	uint	0-2	x	
14		x	-		Max-Age	uint	0-4	*	
15	x	x	-	x	Uri-Query	string	0-255	x	
17	x				Accept	uint	0-2	x	
20				x	Location-Query	string	0-255	x	
23	x	x	-	-	Block2	uint	0-3	*	
27	x	x	-	-	Block1	uint	0-3	*	
28			x		Size2	unit	0-4	*	
35	x	x	-		Proxy-Uri	string	1-1034		*
39	x	x	-		Proxy-Scheme	string	1-255		x
60			x		Size1	uint	0-4	*	

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect, A=Integrity Protect, *=Special

Figure 4: Protection of CoAP Options

A summary of how options are protected and processed is shown in Figure 4. The CoAP options are partitioned into two classes:

- E - options which are encrypted and integrity protected, and

- A - options which are only integrity protected.

Options within each class are protected and processed in a similar way, but certain options which require special processing as described in the subsections and indicated by a '*' in Figure 4.

Unless specified otherwise, CoAP options not listed in Figure 4 SHALL be encrypted and integrity protected and processed as class E options.

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New COAP options SHOULD be of class E and SHOULD NOT have outer options unless a forwarding proxy needs to read an option value. If a certain option is both inner and outer, the

two values SHOULD NOT be the same, unless a proxy is required by specification to be able to read the end-to-end value.

4.3.1. Class E Options

For options in class E (see Figure 4) the option value in the unprotected CoAP message, if present, SHALL be encrypted and integrity protected between the endpoints, and thus is not visible to or possible to change by intermediary nodes. Hence the actions resulting from the use of such options is analogous to communicating in a protected manner with the endpoint. For example, a client using an ETag option will not be served by a proxy.

The sending endpoint SHALL write the class E option from the unprotected CoAP message into the plaintext of the COSE object (see Section 6.2 and Section 6.4).

Except for the special options described in the subsections, the sending endpoint SHALL NOT use the outer options of class E. However, note that an intermediary may, legimitimately or not, add, change or remove the value of an outer option.

Execept for the Block options Section 4.3.1.3, the receiving endpoint SHALL discard any outer options of class E from the protected CoAP message and SHALL replace it with the value from the COSE object when present (see Section 6.3 and Section 6.5).

4.3.1.1. Max-Age

An inner Max-Age option is used as defined in [RFC7252] taking into account that it is not accessible to proxies.

Since OSCOAP binds CoAP responses to requests, a cached response would not be possible to use for any other request. Therefore, there SHOULD be an outer Max-Age option with value zero to prevent caching of responses (see Section 5.6.1 of [RFC7252]).

The outer Max-Age option SHALL NOT be encrypted and SHALL NOT be integrity protected.

4.3.1.2. Observe

The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1.2).

An inner Observe option is used between endpoints. In order for a proxy to support forwarding of notifications, there SHALL be an outer Observe option. To simplify the processing in the server, the outer

option SHOULD have the same value as the inner Observe option. The outer Observe option MAY have different values than the inner, but the order of the different values is SHALL be the same as for the inner Observe option.

The outer Observe option SHALL neither be encrypted nor integrity protected.

4.3.1.3. The Block Options

The Block options (Block1, Block2, Size1 and Size2) MAY be either only inner options, only outer options or both inner and outer options. The inner and outer options are processed independently.

The inner block options are used for endpoint-to-endpoint secure fragmentation of payload into blocks and protection of information about the fragmentation (block number, last block, etc.). Additionally, a proxy may arbitrarily do fragmentation operations on the protected CoAP message, adding outer block options that are not intended to be verified by any endpoint or proxy.

There SHALL be a security policy defining a maximum unfragmented message size for inner Block options such that messages exceeding this size SHALL be fragmented by the sending endpoint.

In addition to the processing defined for the inner Block options inherent to class E options, the AEAD Tag from each block SHALL be included in the calculation of the Tag for the next block (see Section 5.2), so that each block in the order being sent can be verified as it arrives.

The protected CoAP message may be fragmented by the sending endpoint or proxy as defined in [RFC7959], in which case the outer Block options are being used. The outer Block options SHALL neither be encrypted nor integrity protected.

An endpoint receiving a message with an outer Block option SHALL first process this option according to [RFC7959], until all blocks of the protected CoAP message has been received, or the cumulated message size of the exceeds the maximum unfragmented message size. In the latter case the message SHALL be discarded. In the former case, the processing of the protected CoAP message continues as defined in this document (see Section 6.3 and Section 6.5).

If the unprotected CoAP message contains Block options, the receiving endpoint processes this according to {{RFC7959}.

4.3.2. Class A Options

Options in this class are used to support forward proxy operations. Class A options SHALL only have outer values and SHALL NOT be encrypted. In order for the destination endpoint to verify the Uri, class A options SHALL be integrity protected.

Uri-Host, Uri-Port, Proxy-Scheme and Proxy-Uri are class A options. When Uri-Host, Uri-Port, Proxy-Scheme options are present, Proxy-Uri is not used [RFC7252]. Proxy-Uri is processed like the other class A options after a pre-processing step (see Section 4.3.2.1.

Except for Proxi-Uri, the sending endpoint SHALL copy the class A option from the unprotected CoAP message to the protected CoAP message. The class A options are inserted in the AAD of the COSE object (see unencrypted-Uri Section 5.2).

4.3.2.1. Proxy-Uri

Proxy-Uri, when present, is split by OSCOAP into class A options and privacy sensitive class E options, which are processed accordingly. When Proxy-Uri is used in the unprotected CoAP message, Uri-* are not present [RFC7252].

The sending endpoint SHALL first decompose the Proxy-Uri value of the unprotected CoAP message into the unencrypted-Uri (Section 5.2) and Uri-Path/Query options according to section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and SHALL be protected and processed as if obtained from the unprotected CoAP message, see Section 4.3.1.

The value of the Proxy-Uri option of the protected CoAP message SHALL be replaced with unencrypted-Uri and SHALL be protected and processed as a class A option, see Section 4.3.2.

5. The COSE Object

This section defines how to use the COSE format [I-D.ietf-cose-msg] to wrap and protect data in the unprotected CoAP message. OSCOAP uses the COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm.

The AEAD algorithm AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg] is mandatory to implement. For AES-CCM-64-64-128 the length of Sender Key and Recipient Key SHALL be 128 bits, the length of nonce, Sender IV, and Recipient IV SHALL be 7 bytes, and the maximum Sequence Number SHALL be 2^56-1. The nonce is

constructed as described in Section 3.1 of [I-D.ietf-cose-msg], i.e. by padding the Partial IV (Sequence Number) with zeroes and XORing it with the context IV (Sender IV or Recipient IV).

Since OSCOAP only makes use of a single COSE structure, there is no need to explicitly specify the structure, and OSCOAP uses the untagged version of the COSE_Encrypt0 structure (Section 2. of [I-D.ietf-cose-msg]). If the COSE object has a different structure, the recipient MUST reject the message, treating it as malformed.

OSCOAP introduces a new COSE Header Parameter, the Sender Identifier:

sid: This parameter is used to identify the sender of the message. Applications MUST NOT assume that 'sid' values are unique. This is not a security critical field. For this reason, it can be placed in the unprotected headers bucket.

name	label	value type	value registry	description
sid	TBD	bstr		Sender Identifier

Table 1: Additional COSE Header Parameter

We denote by Plaintext the data that is encrypted and integrity protected, and by Additional Authenticated Data (AAD) the data that is integrity protected only, in the COSE object.

The fields of COSE_Encrypt0 structure are defined as follows (see example in Appendix C.4).

- The "Headers" field is formed by:
  - The "protected" field, which SHALL include:
    - The "Partial IV" parameter. The value is set to the Sender Sequence Number. The Partial IV is a byte string (type: bstr), and SHOULD be of minimum length needed to encode the sequence number.
    - The "kid" parameter. The value is set to the Context Identifier (see Section 3). This parameter is optional if the message is a CoAP response.
    - Optionally, the parameter called "sid", defined below. The value is set to the Sender ID (see Section 3). Note that

since this parameter is sent in clear, privacy issues SHOULD be considered by the application defining the Sender ID.

* The "unprotected" field, which SHALL be empty.

- The "ciphertext" field is computed from the Plaintext (see Section 5.1) and the Additional Authenticated Data (AAD) (see Section 5.2) and encoded as a byte string (type: bstr), following Section 5.2 of [I-D.ietf-cose-msg].

5.1. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 5) consisting of:

- all CoAP Options present in the unprotected message which are encrypted (see Section 4), in the order as given by the Option number (each Option with Option Header including delta to previous included encrypted option); and

- the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

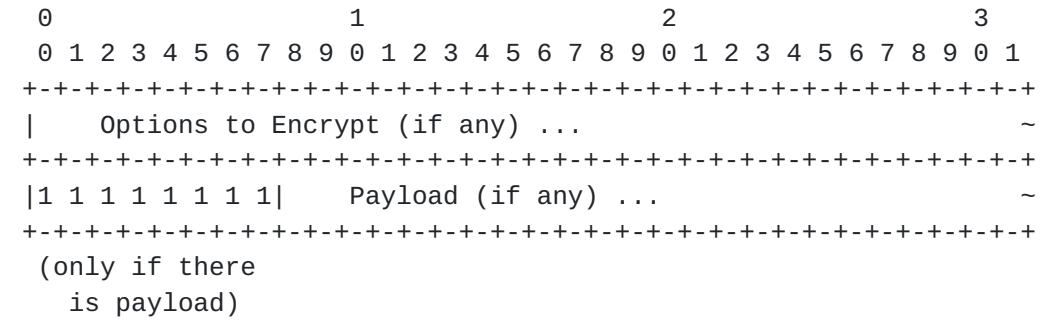


Figure 5: Plaintext

5.2. Additional Authenticated Data

The Additional Authenticated Data ("Enc_structure") as described is Section 5.3 of [I-D.ietf-cose-msg] includes:

- the "context" parameter, which has value "Encrypted"

- the "protected" parameter, which includes the "protected" part of the "Headers" field;

- the "external_aad" is a serialized CBOR array Figure 6 where the exact content is different in requests (external_aad_req) and repsonses (external_aad_resp). It contains:

* ver: uint, contains the CoAP version number, as defined in Section 3 of [RFC7252]

* code: uint, contains is the CoAP Code of the unprotected CoAP message, as defined in Section 3 of [RFC7252].

* alg: int, contains the Algorithm from the security context used for the exchange (see Section 3.1);

* unencrypted-uri: tstr with tag URI, contains the part of the URI which is not encrypted, and is composed of the request scheme (Proxy-Scheme if present), Uri-Host and Uri-Port (if present) options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request;

* cid : bstr, contains the cid for the request (which is same as the cid for the response).

* id : bstr, is the identifier for the endpoint sending the request and verifying the response; which means that for the endpoint sending the response, the id has value Recipient ID, while for the endpoint receiving the response, id has the value Sender ID.

* seq : bstr, is the value of the "Partial IV" in the COSE object of the request (see Section 5).

* tag-previous-block: bstr, contains the AEAD Tag of the message containing the previous block in the sequence, as enumerated by Block1 in the case of a request and Block2 in the case of a response, if the message is fragmented using a block option [RFC7959].

```
external_aad = external_aad_req / external_aad_resp

external_aad_req = [
   ver : uint,
   code : uint,
   alg : int,
   unencrypted-uri : uri,
   ? tag-previous-block : bstr
]

external_aad_resp = [
   ver : uint,
   code : uint,
   alg : int,
   cid : bstr,
   id : bstr,
   seq : bstr,
   ? tag-previous-block : bstr
]
```

Figure 6: External AAD (external_aad)

The encryption process is described in Section 5.3 of [I-D.ietf-cose-msg].

6. Protecting CoAP Messages

6.1. Replay and Freshness Protection

In order to protect from replay of messages and verify freshness, a CoAP endpoint SHALL maintain a Sender Sequence Number and a Recipient Replay Window in the security context. An endpoint uses the Sender Sequence Number to protect messages to send and the Recipient Replay Window to verify received messages, as described in Section 3.

A receiving endpoint SHALL verify that the Sequence Number (Partial IV) received in the COSE object has not been received before in the security context identified by the Cid. The size of the Replay Window depends on the use case and lower protocol layers. In case of reliable and ordered transport, the recipient MAY just store the last received sequence number and require that newly received Sequence Numbers equals the last received Recipient Sequence Number + 1.

The receiving endpoint SHALL reject messages with a sequence number greater than the maximum value of the Partial IV. This maximum value is algorithm specific, for example for AES-CCM-64-64-128 it is 2^56-1.

OSCOAP responses are verified to match a prior request, by including the unique transaction identifier (Tid as defined in Section 3) of the request in the Additional Authenticated Data of the response message. In case of CoAP observe, each notification MUST be verified using the Tid of the observe registration, so the Tid of the registration needs to be cached by the observer until the observation ends.

If a CoAP server receives a request with the Object-Security option, then the server SHALL include the Tid of the request in the AAD of the response, as described in Section 6.4.

If the CoAP client receives a response with the Object-Security option, then the client SHALL verify the integrity of the response, using the Tid of its own associated request in the AAD, as described in Section 6.5.

6.2. Protecting the Request

Given an unprotected CoAP request, including header, options and payload, the client SHALL perform the following steps to create a protected CoAP request using a security context associated with the target resource (see Section 3.2.2).

When using Uri-Host or Proxy-Uri in the construction of the request, the <host> value MUST be a reg-name ([RFC3986]), and not an IP-literal or IPv4address, for canonicalization of the destination address.

1. Compute the COSE object as specified in Section 5

   * the AEAD nonce is created by XORing the Sender IV (context IV) with the Sender Sequence Number (partial IV).

   * If the block option is used, the AAD includes the AEAD Tag from the previous block sent (from the second block and following) Section 5.2. This means that the endpoint MUST store the Tag of each last-sent block to compute the following.

   * Note that the 'sid' field containing the Sender ID is included in the COSE object (Section 5) if the application needs it.

2. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload, based on the unprotected CoAP message:

   * The CoAP header is the same as the unprotected CoAP message.

* If present, the CoAP option Proxy-Uri is decomposed as described in Section 4.3.2.1.

* The CoAP options which are of class E (Section 4) are removed. The Object-Security option is added.

* If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

3. Store the association Token - Cid. The Client SHALL be able to find the correct security context used to protect the request and verify the response with use of the Token of the message exchange.

4. Increment the Sender Sequence Number by one. If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the client MUST NOT process any more requests with the given security context. The client SHOULD acquire a new security context (and consequently inform the server about it) before this happens. The latter is out of scope of this memo.

6.3. Verifying the Request

A CoAP server receiving an unprotected CoAP request to access a protected resource (as defined Section 3.2.2) SHALL reject the message with error code 4.01 (Unauthorized).

A CoAP server receiving a message containing the Object-Security option and a outer Block option SHALL first process this option according to [RFC7959], until all blocks of the protected CoAP message has been received, see Section 4.3.1.3.

A CoAP server receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Context Identifier in the "kid" parameter in the received COSE object:

1. Verify the Sequence Number in the Partial IV parameter, as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the server MUST stop processing the request.

2. Recreate the Additional Authenticated Data, as described in Section 5.

* If the block option is used, the AAD includes the AEAD Tag from the previous block received (from the second block and following) Section 5.2. This means that the endpoint MUST store the Tag of each last-received block to compute the following.

* Note that the server's <host> value MUST be a reg-name ([RFC3986]), and not an IP-literal or IPv4address.

3. Compose the AEAD nonce by XORing the Recipient IV (context IV) with the padded Partial IV parameter, received in the COSE Object.

4. Retrieve the Recipient Key.

5. Verify and decrypt the message. If the verification fails, the server MUST stop processing the request.

6. If the message verifies, update the Recipient Replay Window, as described in Section 6.1.

7. Restore the unprotected request by adding any decrypted options or payload from the plaintext. Any outer E options (Section 4) are overwritten. The Object-Security option is removed.

6.4. Protecting the Response

A server receiving a valid request with a protected CoAP message (i.e. containing an Object-Security option) SHALL respond with a protected CoAP message.

Given an unprotected CoAP response, including header, options, and payload, the server SHALL perform the following steps to create a protected CoAP response, using the security context identified by the Context Identifier of the received request:

1. Compute the COSE object as specified in Section Section 5

* The AEAD nonce is created by XORing the Sender IV (context IV) and the padded Sender Sequence Number.

* If the block option [RFC7959] is used, the AAD includes the AEAD Tag from the previous block sent (from the second block and following) Section 5.2. This means that the endpoint MUST store the Tag of each last-sent block to compute the following. Note that this applies even for random access of blocks, i.e. when blocks are not requested in the order of their relative number (NUM).

2. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload based on the unprotected CoAP message:

   * The CoAP header is the same as the unprotected CoAP message.

   * The CoAP options which are of class E are removed, except any special option (labelled '*') that is present which has its outer value (Section 4). The Object-Security option is added.

   * If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

3. Increment the Sender Sequence Number by one. If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the server MUST NOT process any more responses with the given security context. The server SHOULD acquire a new security context (and consequently inform the client about it) before this happens. The latter is out of scope of this memo.

Note the differences between generating a protected request, and a protected response, for example whether "kid" is present in the header, or whether Destination URI or Tid is present in the AAD, of the COSE object.

6.5. Verifying the Response

A CoAP client receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Token of the received response:

1. If the message contain an outer Block option the client SHALL process this option according to [RFC7959], until all blocks of the protected CoAP message has been received, see Section 4.3.1.3.

2. Verify the Sequence Number in the Partial IV parameter as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the client MUST stop processing the response.

3. Recreate the Additional Authenticated Data as described in Section 5.

* If the block option is used, the AAD includes the AEAD Tag from the previous block received (from the second block and following) Section 5.2. This means that the endpoint MUST store the Tag of each last-received block to compute the following.

4. Compose the AEAD nonce by XORing the Recipient IV (context IV) with the Partial IV parameter, received in the COSE Object.

5. Retrieve the Recipient Key.

6. Verify and decrypt the message. If the verification fails, the client MUST stop processing the response.

7. If the message verifies, update the Recipient Replay Window, as described in Section 6.1.

8. Restore the unprotected response by adding any decrypted options or payload from the plaintext. Any class E options (Section 4) are overwritten. The Object-Security option is removed.

7. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as DTLS only protects data hop-by-hop. As a consequence the intermediary nodes can read and modify information. The trust model where all intermediate 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 brittle.

DTLS protects hop-by-hop the entire CoAP message, including header, options, and payload. OSCOAP protects end-to-end the payload, and all information in the options and header, that is not required for forwarding (see Section 4). DTLS and OSCOAP can be combined, thereby enabling 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.

The CoAP message layer, however, cannot be protected end-to-end through intermediary devices since the parameters Type and Message ID, as well as Token and Token Length may be changed by a proxy. Moreover, messages that are not possible to verify should for security reasons not always be acknowledged but in some cases be

silently dropped. This would not comply with CoAP message layer, but does not have an impact on the application layer security solution, since message layer is excluded from that.

The use of COSE to protect CoAP messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common shared secret material and key derivation function in client and server. EDHOC [I-D.selander-ace-cose-ecdhe] describes an augmented Diffie-Hellman key exchange to produce forward secret keying material and agree on crypto algorithms necessary for OSCOAP, authenticated with pre-established credentials. These pre-established credentials may, in turn, be provisioned using a trusted third party such as described in the OAuth-based ACE framework [I-D.ietf-ace-oauth-authz]. An OSCOAP profile of ACE is described in [I-D.seitz-ace-oscoap-profile].

The mandatory-to-implement AEAD algorithm AES-CCM-64-64-128 is selected for broad applicability in terms of message size (2^64 blocks) and maximum no. messages (2^56-1). Compatibility with CCM* is achieved by using the algorithm AES-CCM-16-64-128 [I-D.ietf-cose-msg].

Most AEAD algorithms require a unique nonce for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. The alternatives to sequence numbers have their issues: very constrained devices may not be able to support accurate time, or to 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.

The encrypted block options enable the sender to split large messages into protected blocks such that the receiving node can verify blocks before having received the complete message. In order to protect from attacks replacing blocks from a different message with the same block number between same endpoints and same resource at roughly the same time, the AEAD Tag from the message containing one block is included in the external_aad of the message containing the next block.

The unencrypted block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the encrypted options allow for

secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

Applications need to use a padding scheme if the content of a message can be determined solely from the length of the payload. As an example, the strings "YES" and "NO" even if encrypted can be distinguished from each other as there is no padding supplied by the current set of encryption algorithms. Some information can be determined even from looking at boundary conditions. An example of this would be returning an integer between 0 and 100 where lengths of 1, 2 and 3 will provide information about where in the range things are. Three different methods to deal with this are: 1) ensure that all messages are the same length. For example using 0 and 1 instead of 'yes' and 'no'. 2) Use a character which is not part of the responses to pad to a fixed length. For example, pad with a space to three characters. 3) Use the PKCS #7 style padding scheme where m bytes are appended each having the value of m. For example, appending a 0 to "YES" and two 1's to "NO". This style of padding means that all values need to be padded.

8. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCOAP. End-to-end integrity protection and encryption of CoAP payload and all options that are not used for forwarding, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

CoAP headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis.

9. IANA Considerations

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

9.1. CoAP Option Numbers Registry

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

9.2. COSE Header Parameters Registry

The "sid" parameter is added to the COSE Header Parameter Registry:

name	label	value type	value registry	description
sid	TBD	bstr		Sender Identifier

9.3. Media Type Registrations

The "application/oscon" media type is added to the Media Types registry:

Type name: application

Subtype name: oscon

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Appendix C of [[this document]].

Interoperability considerations: N/A

Published specification: [[this document]]

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

* 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: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

9.4. CoAP Content Format Registration

The "application/oscon" content format is added to the CoAP Content Format registry:

Media type	Encoding	ID	Reference
application/oscon	-	70	[[this document]]

10. Acknowledgments

The following individuals provided input to this document: Carsten Bormann, Joakim Brorsson, Martin Gunnarsson, Klaus Hartke, Jim Schaad, Marco Tiloca, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

11. References

11.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

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

11.2. Informative References

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-04 (work in progress), October 2016.

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-01 (work in progress), July 2016.

[I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.

[I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-04 (work in progress), October 2016.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-05 (work in progress), October 2016.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-09 (work in progress), September 2016.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

Appendix A. Overhead

OSCOAP transforms an unprotected CoAP message to a protected CoAP message, and the protected CoAP message is larger than the unprotected CoAP message. This appendix illustrates the message expansion.

A.1. Length of the Object-Security Option

The protected CoAP message contains the COSE object. The COSE object is included in the payload if the message type of the unprotected CoAP message allows payload or else in the Object-Security option. In the former case the Object-Security option is empty. So the length of the Object-Security option is either zero or the size of the COSE object, depending on whether the CoAP message allows payload or not.

Length of Object-Security option = { 0, size of COSE Object }

A.2. Size of the COSE Object

The size of the COSE object is the sum of the sizes of

o the Header parameters,

- the Cipher Text (excluding the Tag),
- the Tag, and
- data incurred by the COSE format itself (including CBOR encoding).

Let's analyze the contributions one at a time:

- The header parameters of the COSE object are the Context Identifier (Cid) and the Sequence Number (Seq) (also part of the Transaction Identifier (Tid)) if the message is a request, and Seq only if the message is a response (see Section 5).
  - The size of Cid is recommended to be 64 bits, but may be shorter, as discussed in Section 3.2.2
  - The size of Seq is variable, and increases with the number of messages exchanged.
  - As the AEAD nonce is generated from the padded Sequence Number and a previously agreed upon context IV it is not required to send the whole nonce in the message.
- The Cipher Text, excluding the Tag, is the encryption of the payload and the encrypted options Section 4, which are present in the unprotected CoAP message.
- The size of the Tag depends on the Algorithm. For example, for the algorithm AES-CCM-64-64-128, the Tag is 8 bytes.
- The overhead from the COSE format itself depends on the sizes of the previous fields, and is of the order of 10 bytes.

A.3. Message Expansion

The message expansion is not the size of the COSE object. The ciphertext in the COSE object is encrypted payload and options of the unprotected CoAP message - the plaintext of which is removed from the protected CoAP message. Since the size of the ciphertext is the same as the corresponding plaintext, there is no message expansion due to encryption; payload and options are just represented in a different way in the protected CoAP message:

- The encrypted payload is in the payload of the protected CoAP message
- The encrypted options are in the Object-Security option or within the payload.

Therefore the OSCOAP message expansion is due to Cid (if present), Seq, Tag, and COSE overhead:

```
Message Overhead = Cid + Seq + Tag + COSE Overhead
```

Figure 7: OSCOAP message expansion

A.4. Example

This section gives an example of message expansion in a request with OSCOAP.

In this example we assume an 8-byte Cid.

- Cid: 0xa1534e3c9cecad84

In the example the sequence number is 225, requiring 1 byte to encode. (The size of Seq could be larger depending on how many messages that has been sent as is discussed in Appendix A.2.)

- Seq: 225

The example is based on AES-CCM-64-64-128.

- Tag is 8 bytes

The COSE object is represented in Figure 8 using CBOR's diagnostic notation.

```
[
  h'a20448a1534e3c9cecad840641e2', / protected:
                                     {04:h'a1534e3c9cecad84',
                                      06:h'e2'} /
  {},                              / unprotected: - /
  Ciph + Tag                       / ciphertext + 8 byte
                                     authentication tag /
]
```

Figure 8: Example of message expansion

Note that the encrypted CoAP options and payload are omitted since we target the message expansion (see Appendix A.3). Therefore the size of the COSE Cipher Text equals the size of the Tag, which is 8 bytes.

The COSE object encodes to a total size of 26 bytes, which is the message expansion in this example. The COSE overhead in this example is 26 - (8 + 1 + 8) = 9 bytes, according to the formula in Figure 7.

Note that in this example two bytes in the COSE overhead are used to encode the length of Cid and the length of Seq.

Figure 9 summarizes these results.

Cid	Seq	Tag	COSE OH	Message OH
8 bytes	1 byte	8 bytes	9 bytes	22 bytes

Figure 9: Message overhead for a 8-byte Cid, 1-byte Seq and 8-byte Tag.

Appendix B. Examples

This section gives examples of OSCOAP. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the COSE message format.

B.1. Secure Access to Sensor

Here is an example targeting the scenario in the Section 2.2.1. - Forwarding of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting the alarm status from a server. In the request, CoAP option Uri-Path is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected (see Section 4). In the response, the CoAP Payload is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected.

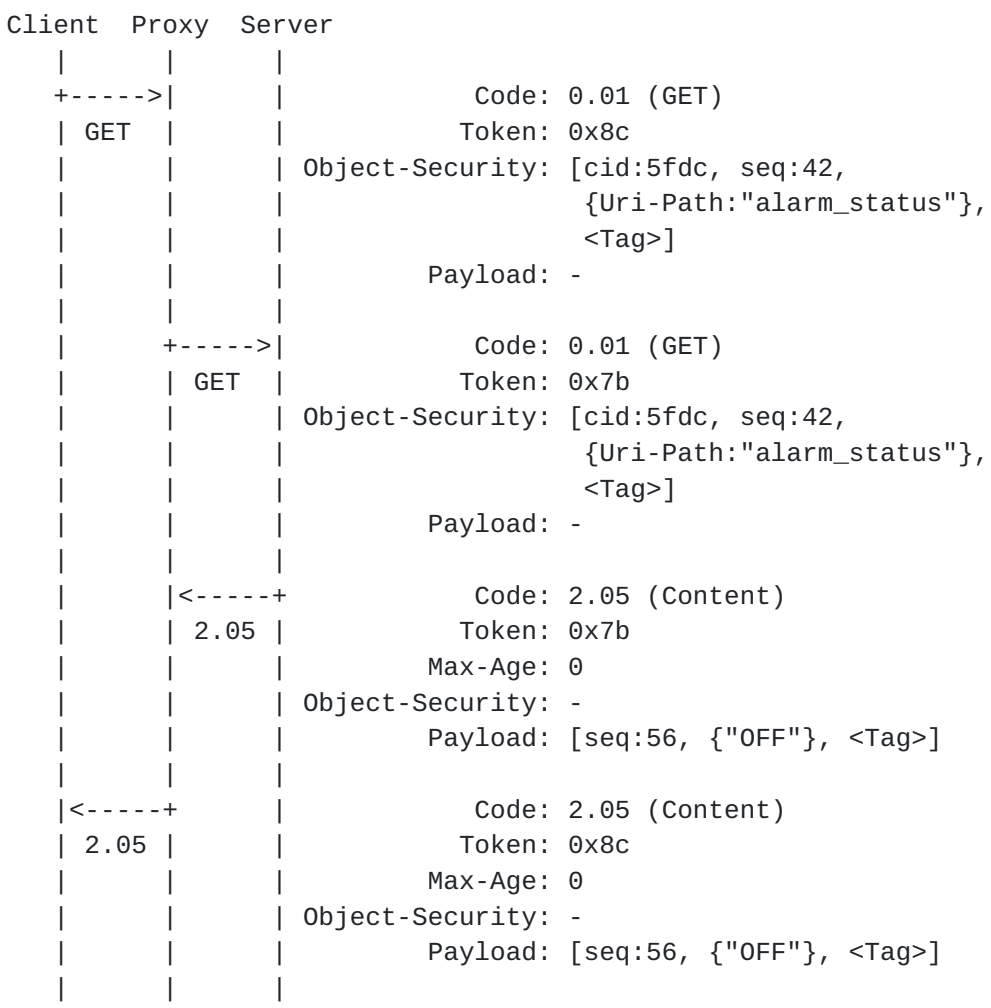


Figure 10: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

Since the unprotected request message (GET) has no payload, the Object-Security option carries the COSE object as its value. Since the unprotected response message (Content) has payload ("OFF"), the COSE object (indicated with [...]) is carried as the CoAP payload.

The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

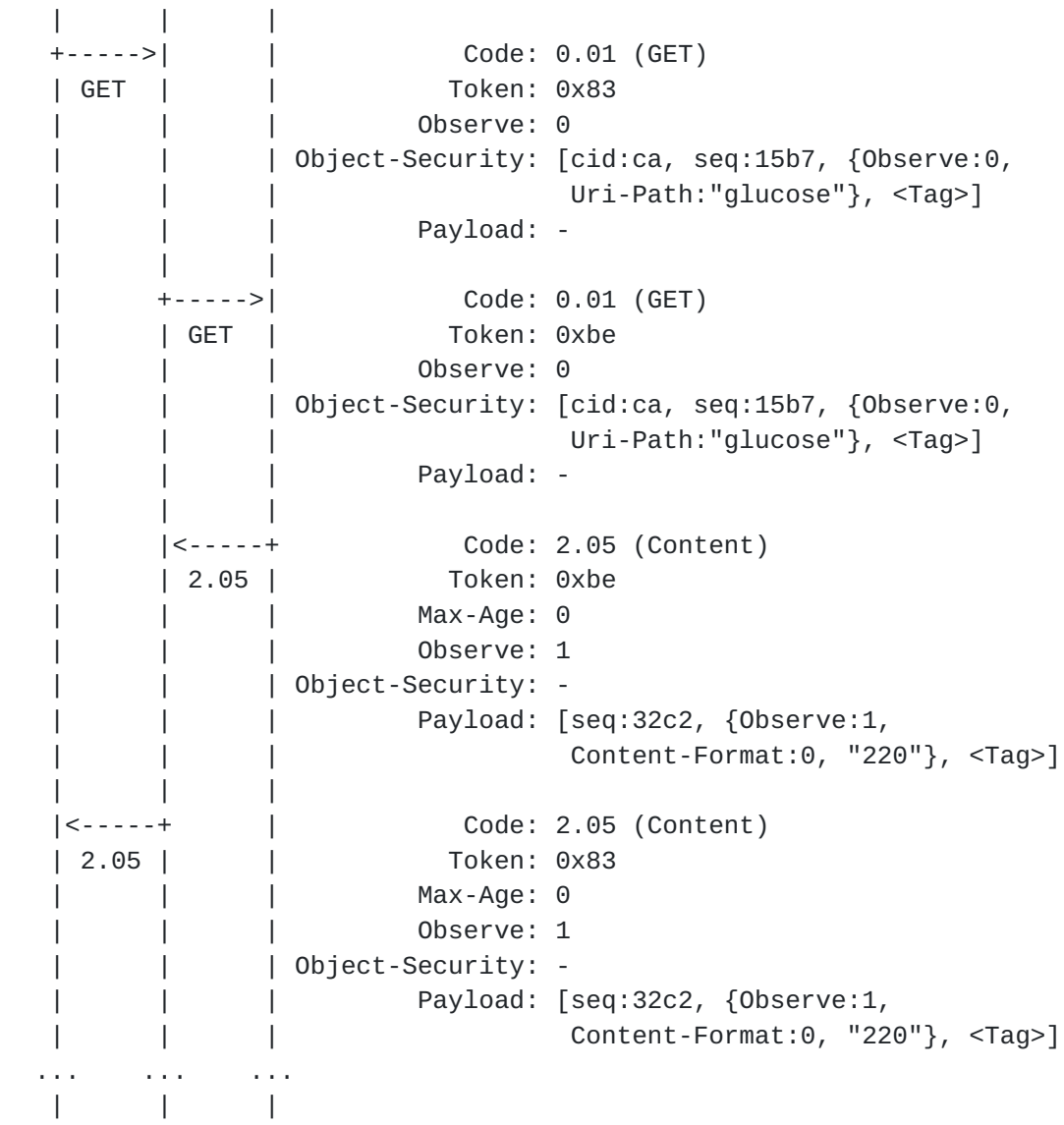
The option Uri-Path (alarm_status) and payload ("OFF") are formatted as indicated in Section 5, and encrypted in the COSE Cipher Text (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence

Number has not been received before and that the response message is generated as a response to the sent request message (see Section 6.1).

B.2. Secure Subscribe to Sensor

Here is an example targeting the scenario in the Forwarding with observe case of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), and first receiving the value 220 mg/dl, and then a second reading with value 180 mg/dl. The CoAP options Observe, Uri-Path, Content-Format, and Payload are encrypted and integrity protected, and the CoAP header field Code is integrity protected (see Section 4).



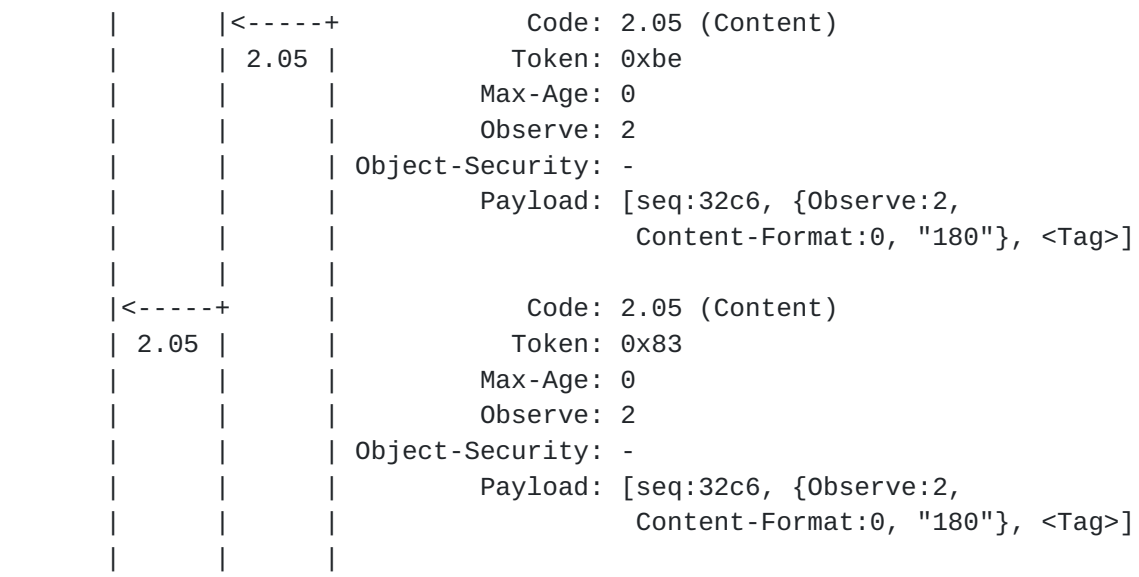


Figure 11: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

Since the unprotected request message (GET) allows no payload, the COSE object (indicated with [...]) is carried in the Object-Security option value. Since the unprotected response message (Content) has payload, the Object-Security option is empty, and the COSE object is carried as the payload.

The COSE header of the request contains a Context Identifier (cid:ca), indicating which security context was used to protect the message and a Sequence Number (seq:15b7).

The options Observe, Content-Format and the payload are formatted as indicated in Section 5, and encrypted in the COSE ciphertext (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence Number has not been received before and that the response message is generated as a response to the subscribe request.

Appendix C. Object Security of Content (OSCON)

OSCOAP protects message exchanges end-to-end between a certain client and a certain server, targeting the security requirements for forward proxy of [I-D.hartke-core-e2e-security-reqs]. In contrast, many use cases require one and the same message to be protected for, and verified by, multiple endpoints, see caching proxy section of [I-D.hartke-core-e2e-security-reqs]. Those security requirements can

be addressed by protecting essentially the payload/content of individual messages using the COSE format ([I-D.ietf-cose-msg]), rather than the entire request/response message exchange. This is referred to as Object Security of Content (OSCON).

OSCON transforms an unprotected CoAP message into a protected CoAP message in the following way: the payload of the unprotected CoAP message is wrapped by a COSE object, which replaces the payload of the unprotected CoAP message. We call the result the "protected" CoAP message.

The unprotected payload shall be the plaintext/payload of the COSE object. The 'protected' field of the COSE object 'Headers' shall include the context identifier, both for requests and responses. If the unprotected CoAP message includes a Content-Format option, then the COSE object shall include a protected 'content type' field, whose value is set to the unprotected message Content-Format value. The Content-Format option of the protected CoAP message shall be replaced with "application/oscon" (Section 9)

The COSE object shall be protected (encrypted) and verified (decrypted) as described in ([I-D.ietf-cose-msg]).

Most AEAD algorithms require a unique nonce for each message. Sequence numbers for partial IV as specified for OSCOAP may be used for replay protection as described in Section 6.1. The use of time stamps in the COSE header parameter 'operation time' [I-D.ietf-cose-msg] for freshness may be used.

OSCON shall not be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected or encrypted. OSCON shall not be used in cases which require a secure binding between request and response.

The scenarios in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs] assume multiple recipients for a particular content. In this case the use of symmetric keys does not provide data origin authentication. Therefore the COSE object should in general be protected with a digital signature.

C.1. Overhead OSCON

In general there are four different kinds of modes that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The use of digital signature is necessary for applications with many legitimate recipients of a given message, and where data origin authentication is required.

To distinguish between these different cases, the tagged structures of COSE are used (see Section 2 of [I-D.ietf-cose-msg]).

The sizes of COSE messages for selected algorithms are detailed in this section.

The size of the header is shown separately from the size of the MAC/ signature. A 4-byte Context Identifier and a 1-byte Sequence Number are used throughout all examples, with these values:

- Cid: 0xa1534e3c

- Seq: 0xa3

For each scheme, we indicate the fixed length of these two parameters ("Cid+Seq" column) and of the Tag ("MAC"/"SIG"/"TAG"). The "Message OH" column shows the total expansions of the CoAP message size, while the "COSE OH" column is calculated from the previous columns following the formula in Figure 7.

Overhead incurring from CBOR encoding is also included in the COSE overhead count.

To make it easier to read, COSE objects are represented using CBOR's diagnostic notation rather than a binary dump.

C.2. MAC Only

This example is based on HMAC-SHA256, with truncation to 8 bytes (HMAC 256/64).

Since the key is implicitly known by the recipient, the COSE_Mac0_Tagged structure is used (Section 6.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
996(                          # COSE_Mac0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                       # unprotected
    h'',                      # payload
    MAC                       # truncated 8-byte MAC
  ]
)
```

This COSE object encodes to a total size of 26 bytes.

Figure 12 summarizes these results.

Structure	Tid	MAC	COSE OH	Message OH
COSE_Mac0_Tagged	5 B	8 B	13 B	26 B

Figure 12: Message overhead for a 5-byte Tid using HMAC 256/64

C.3. Signature Only

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

Since only one signature is used, the COSE_Sign1_Tagged structure is used (Section 4.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
997(                          # COSE_Sign1_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                      # unprotected
    h'',                     # payload
    SIG                      # 64-byte signature
  ]
)
```

This COSE object encodes to a total size of 83 bytes.

Figure 13 summarizes these results.

Structure	Tid	SIG	COSE OH	Message OH
COSE_Sign1_Tagged	5 B	64 B	14 B	83 bytes

Figure 13: Message overhead for a 5-byte Tid using 64 byte ECDSA signature.

C.4. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-CCM with the Tag truncated to 8 bytes.

Since the key is implicitly known by the recipient, the COSE_Encrypt0_Tagged structure is used (Section 5.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
  993(                        # COSE_Encrypt0_Tagged
    [
      h'a20444a1534e3c0641a3', # protected:
                                 {04:h'a1534e3c',
                                  06:h'a3'}
      {},                     # unprotected
      TAG                     # ciphertext + truncated 8-byte TAG
    ]
  )
```

This COSE object encodes to a total size of 25 bytes.

Figure 14 summarizes these results.

Structure	Tid	TAG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	12 B	25 bytes

Figure 14: Message overhead for a 5-byte Tid using AES_128_CCM_8.

C.5. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-CCM and ECDSA with 64 bytes signature. The same assumption on the security context as in Appendix C.4. COSE defines the field 'counter signature w/o headers' that is used here to sign a COSE_Encrypt0_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
 993(                        # COSE_Encrypt0_Tagged
   [
     h'a20444a1534e3c0641a3', # protected:
                                {04:h'a1534e3c',
                                 06:h'a3'}
     {9:SIG},                 # unprotected:
                                 09: 64 bytes signature
     TAG                      # ciphertext + truncated 8-byte TAG
   ]
 )
```

This COSE object encodes to a total size of 92 bytes.

Figure 15 summarizes these results.

Structure	Tid	TAG	SIG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	64 B	15 B	92 B

Figure 15: Message overhead for a 5-byte Tid using AES-CCM countersigned with ECDSA.

Authors' Addresses

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig@sics.se