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

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

October 19, 2016

Object Security of CoAP (OSCOAP)
draft-ietf-core-object-security-00

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 April 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, and freshness of requests and responses. It 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].

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

Two different scopes of object security are defined:

- OSCOAP = object security of CoAP, signaled with the Object-Security option.

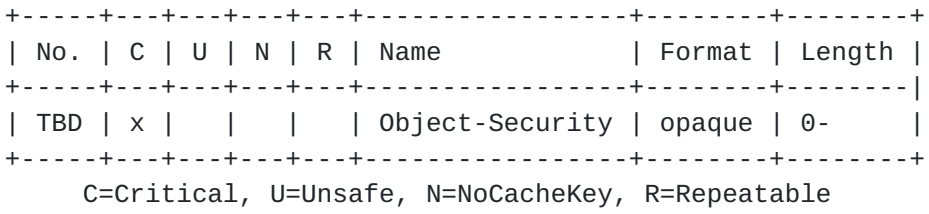
- OSCON = object security of content, signaled with Content Format/ Media Type set to application/oscon (defined in Appendix C).

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 has 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 has 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.

- If the unprotected message does not have 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.

More details about the message overhead caused by the Object-Security option is 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 establish a security context in client and server based on common shared secret material and a key derivation function (KDF).

The EDHOC protocol [I-D.selander-ace-cose-ecdhe] enables the establishment of secret material 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.

3.1. 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 Common Context includes common security material. The endpoint protects the messages sent using the Sender Context. The endpoint verifies the messages received using the Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Note that, because of that, the two security contexts identified by the same Context Identifiers in the two endpoints are not the same, but they are partly mirrored.

An example is shown in Figure 3.

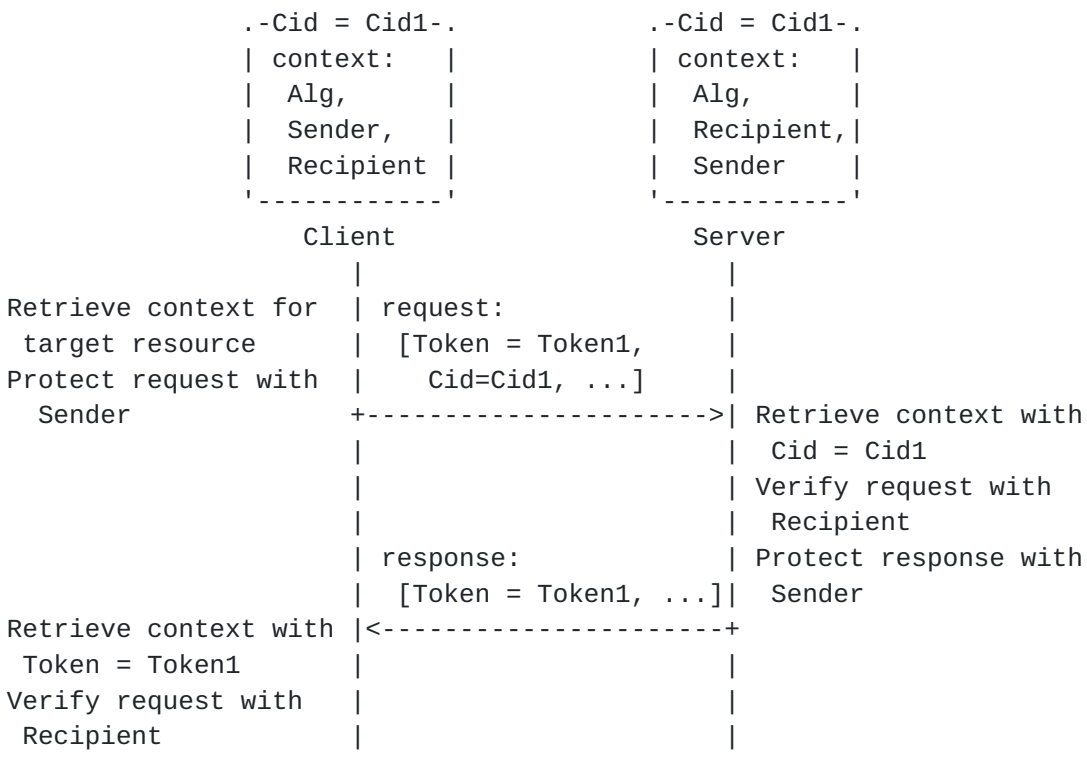


Figure 3: Retrieval and use of the Security Context

The Common Context structure 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 (base_key). Byte string containing the key used to derive the security context Section 3.2.

The Sender Context structure contains the following parameters:

- Sender ID. Variable length byte string identifying oneself. 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 portion of IV (context IV in [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, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.

The Recipient Context structure contains the following parameters:

- Recipient ID. Variable length byte string identifying the endpoint messages are received from or sent to. 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 Sequence Number. Non-negative integer enumerating the COSE objects received, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.

- Replay Window. The replay protection window for messages received, equivalent to the functionality described in Section 4.1.2.6 of [RFC6347].

The 3-tuple (Cid, Sender ID, Sender Sequence Number) is called Transaction Identifier (Tid), and SHALL be unique for each COSE object and server. 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 the challenge becomes signed by the server.

The client and server may change roles while maintaining the same security context. The former server will then make the request using the Sender Context, the former client will verify the request using its Recipient Context etc.

3.2. Security Context Establishment

This section aims at describing how to establish the security context, given some input parameters. The input parameters, which are established in a previous phase, are:

- Context Identifier (Cid)
- Algorithm (Alg)
- Base Key (base_key)
- Sender ID
- Recipient ID
- Replay Window (optionally)

These are included unchanged in the security context. We give below some indications on how applications should select these parameters. Moreover, the following parameters are established as described below:

- Sender Key
- Sender IV
- Sender Sequence Number
- Recipient Key
- Recipient IV
- Recipient Sequence Number
- Replay Window

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

Given a common shared secret material and a common key derivation function, the client and server can derive the security context necessary to run OSCOAP. The derivation procedure described here MUST NOT be executed more than once on a set of common secret material. Also, the same base_key SHOULD NOT be used in different security contexts (identified by different Cids).

The procedure assumes that the common shared secret material is uniformly random and that the key derivation function is HKDF

[RFC5869]. This is for example the case after having used EDHOC [I-D.selander-ace-cose-ecdhe].

Assumptions:

- The hash function, denoted HKDF, is the HMAC based key derivation function defined in [RFC5869] with specified hash function

- The common shared secret material, denoted base_key, is uniformly pseudo-random of length at least equal to the output of the specified hash function

The security context parameters Sender Key/IV, Recipient Key/IV SHALL be derived using the HKDF-Expand primitive [RFC5869]:

output parameter = HKDF-Expand(base_key, info, key_length),

where:

- base_key is defined above

- info = Cid || Sender ID/Recipient ID || "IV"/"Key" || Algorithm || key_length

- key_length is the key size of the AEAD algorithm

The Sender/Recipient Key shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "Key", the Algorithm and the key_length. The Sender/Recipient IV shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "IV", the Algorithm and the key_length.

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

3.2.2. Sequence Numbers and Replay Window

The values of the Sequence Numbers are initialized to 0 during establishment of the security context. The default Replay Window size of 64 is used if no input parameter is provided in the set up phase.

3.2.3. Context Identifier and Sender/Recipient ID

As mentioned, Cid, Sender ID and Recipient ID are established in a previous phase. How this is done is application specific, but some guidelines are given in this section.

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.

In the same phase during which the Cid is established in the endpoint, the application informs the endpoint what resource can be accessed using the corresponding security context. The granularity of that is decided by the application (resource, host, etc). The endpoint SHALL save the association resource-Cid, in order to be able to retrieve the correct security context to access a resource.

The Sender ID and Recipient ID are also established in the endpoint during the previous set up phase. The application SHOULD make sure that these identifiers are locally unique in the set of all endpoints using the same security context. If it is not the case, it is again the role of the application to specify how to handle collisions.

In case of EDHOC [I-D.selander-ace-cose-ecdhe]) the Cid is the hash of the messages exchanged.

4. Protected CoAP Message Fields

This section defines how the 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].

The CoAP Payload SHALL be encrypted and integrity protected.

The CoAP Header fields Version and Code SHALL be integrity protected but not encrypted. The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length SHALL neither be integrity protected nor encrypted.

Protection of CoAP Options can be summarized as follows:

- To prevent information leakage, Uri-Path and Uri-Query SHALL be encrypted. If Proxy-Uri is used and thus Uri-* are not present, then OSCOAP implementation MUST first split the Proxy-Uri into the unencrypted Uri Section 5.2 and the Uri-Path/Query options (according to section 6.4 of [RFC7252]), replace the Proxy-Uri value with the unencrypted Uri, and encrypt Uri-Path/Query, which will then be carried in the ciphertext. This means that the proxy will not be able to see that Uri-Path and Uri-Query options are present in the message and will thus process the message as indicated by CoAP.

- The CoAP Options Uri-Host, Uri-Port, Proxy-Uri, and Proxy-Scheme SHALL neither be encrypted, nor integrity protected. Note that even though these options are not protected, their values are included in the additional authenticated data, thus they are indirectly integrity protected (cf. protection of the unencrypted Uri in Section 5.2).

- The other CoAP options SHALL be encrypted and integrity protected.

A summary of which options are encrypted or integrity protected is shown in Figure 4.

No.	C	U	N	R	Name	Format	Length	E	D
1	x			x	If-Match	opaque	0-8	x	
3	x	x	-		Uri-Host	string	1-255		
4				x	ETag	opaque	1-8	x	
5	x				If-None-Match	empty	0	x	
6		x	-		Observe	uint	0-3	x	x
7	x	x	-		Uri-Port	uint	0-2		
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	x	x
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	x	x
27	x	x	-	-	Block1	uint	0-3	x	x
28			x		Size2	unit	0-4	x	x
35	x	x	-		Proxy-Uri	string	1-1034		
39	x	x	-		Proxy-Scheme	string	1-255		
60			x		Size1	uint	0-4	x	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect, D=Duplicate.

Figure 4: Protection of CoAP Options

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

The encrypted options are in general omitted from the protected CoAP message and not visible to intermediary nodes (see Section 6.2 and Section 6.4). Hence the actions resulting from the use of corresponding options is analogous to the case of communicating

directly with the endpoint. For example, a client using an ETag option will not be served by a proxy.

However, some options which are encrypted need to be readable in the protected CoAP message to support certain proxy functions. A CoAP option which may be both encrypted in the COSE object of the protected CoAP message, and also unencrypted as CoAP option in the protected CoAP message, is called "duplicate". The "encrypted" value of a duplicate option is intended for the destination endpoint and the "unencrypted" value is intended for a proxy. The unencrypted value is not integrity protected.

- The Max-Age option is duplicate. The unencrypted Max-Age SHOULD have value zero to prevent caching of responses. The encrypted Max-Age is used as defined in [RFC7252] taking into account that it is not accessible to proxies.

- The Observe option is duplicate. If Observe is used, then the encrypted Observe and the unencrypted Observe SHALL have the same value. The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1.2).

- The block options Block1 and Block2 are duplicate. The encrypted block options is used for end-to-end secure fragmentation of payload into blocks and protected information about the fragmentation (block number, last block, etc.). The MAC from each block is included in the calculation of the MAC for the next block's (see Section 5.2). In this way, each block in ordered sequence from the first block can be verified as it arrives. The unencrypted block option allows for arbitrary proxy fragmentation operations which cannot be verified by the endpoints. An intermediary node can generate an arbitrarily long sequence of blocks. However, since it is possible to protect fragmentation of large messages, there SHALL be a security policy defining a maximum unfragmented message size such that messages exceeding this size SHALL be fragmented by the sending endpoint. Hence an endpoint receiving fragments of a message that exceeds maximum message size SHALL discard this message.

- The size options Size1 and Size2 are duplicate, analogously to the block options.

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New COAP options SHALL be encrypted and integrity protected. New COAP options SHOULD NOT be duplicate unless a forwarding proxy needs to read the option. If an option is registered as duplicate, the duplicate value SHOULD NOT be the same

as the end-to-end value, unless the proxy is required by specification to be able to read the end-to-end value.

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 mandatory to support AEAD algorithm is AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg]. For AES-CCM-64-64-128 the length of Sender Key and Recipient Key SHALL be 128 bits, the length of IV, Sender IV, and Recipient IV SHALL be 7 bytes, and the maximum Sender Sequence Number and Recipient Sequence Number SHALL be 2^56-1. The IV is constructed using a Partial IV exactly like in Section 3.1 of [I-D.ietf-cose-msg], i.e. by padding the Sender Sequence Number or the Recipient Sequence Number with zeroes and XORing it with the Sender IV or Recipient IV, respectively.

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.

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), where the length is the minimum length needed to encode the sequence number. An Endpoint that receives a COSE object with a sequence number encoded with leading zeroes (i.e. longer than the minimum needed length) SHALL reject the corresponding message as malformed.
    - 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.

o The "cipher text" 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].

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

5.1. Plaintext

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

o 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

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

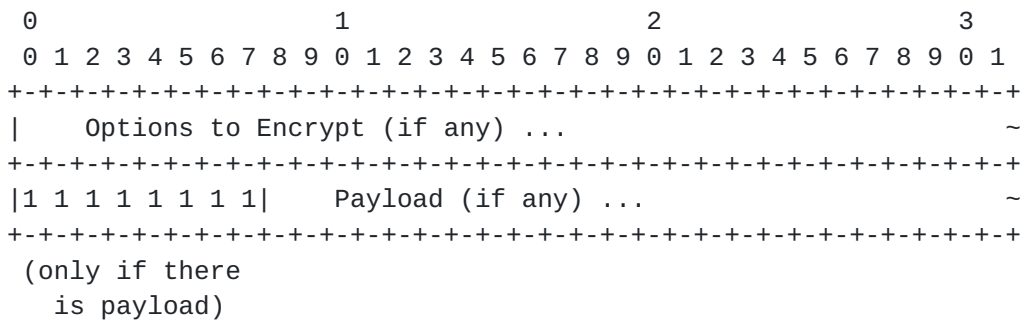


Figure 5: Plaintext

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 (see Figure 8) that contains, in the given order:
  - ver: uint, contains the CoAP version number of the unprotected CoAP message, as defined in Section 3 of [RFC7252]
  - code: bstr, contains is the CoAP Code of the unprotected CoAP message, as defined in Section 3 of [RFC7252].
  - alg: bstr, contains the serialized Algorithm from the security context used for the exchange (see Section 3.1);
  - unencrypted-uri: tstr, 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 options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request;
  - transaction-id: bstr, only included if the message to protect or verify is a CoAP response, contains the Transaction Identifier (Tid) of the associated CoAP request (see Section 3). Note that the Tid is the 3-tuple (Cid, Sender ID, Sender Sequence Number) for the endpoint sending the request and verifying the response; which means that for the endpoint sending the response, the Tid has value (Cid, Recipient ID,

seq), where seq is the value of the "Partial IV" in the COSE object of the request (see Section 5); and

* mac-previous-block: bstr, contains the MAC 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_req = [
    ver : uint,
    code : bstr,
    alg : bstr,
    unencrypted-uri : tstr,
    ? mac-previous-block : bstr
]
```

Figure 6: external_aad for a request

```
external_aad_resp = [
    ver : uint,
    code : bstr,
    alg : bstr,
    transaction-id : bstr,
    ? mac-previous-block : bstr
]
```

Figure 7: external_aad for a response

```
external_aad = external_aad_req / external_aad_resp
```

Figure 8: 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 Sequence Number associated to a security context, which is identified with a Context Identifier (Cid). The two sequence numbers are the highest sequence number the endpoint has sent and the highest sequence number the endpoint has received. An endpoint uses the Sender Sequence Number to protect messages to send and the Recipient

Sequence Number to verify received messages, as described in Section 3.

Depending on use case and ordering of messages provided by underlying layers, an endpoint MAY maintain a sliding replay window for Sequence Numbers of received messages associated to each Cid. In case of reliable transport, the receiving endpoint MAY require that the Sequence Number of a received message equals last Sequence Number + 1.

A receiving endpoint SHALL verify that the Sequence Number received in the COSE object has not been received before in the security context identified by the Cid. The receiving endpoint SHALL also reject messages with a sequence number greater than 2^56-1.

OSCOAP is a challenge-response protocol, where the response is 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.

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 3.2.3).

1. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the client MUST NOT process any 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.

2. Compute the COSE object as specified in Section 5

   * the IV in the AEAD 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 MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-sent fragment 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.

3. 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 encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. 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.

4. Store in memory 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.

6.3. Verifying the Request

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 MAC from the previous fragment received (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-received fragment to compute the following.

3. Compose the IV by XORing the Recipient IV (context IV) with the 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 Sequence Number or Replay Window, as described in Section 6.1.

7. Restore the unprotected request by adding any decrypted options or payload from the plaintext. Any duplicate 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. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). 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.

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

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

   * If the block option is used, the AAD includes the MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST

store the MAC of each last-sent fragment to compute the following.

3. 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 encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. 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.

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. 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.

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

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

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

4. Retrieve the Recipient Key.

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

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

7. Restore the unprotected response by adding any decrypted options or payload from the plaintext. Any duplicate 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].

For symmetric encryption it is required to have a unique IV for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. The context IVs (Sender IV and Recipient IV) SHOULD be established between sender and recipient before the message is sent, for example using the method in [I-D.selander-ace-cose-ecdhe], to avoid the overhead of sending it in each message.

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). For 128 bit CCM*, use instead AES-CCM-16-64-128 [I-D.ietf-cose-msg].

If the recipient accepts any sequence number larger than the one previously received (less than the maximum sequence number), 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 IVs. 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 fragments such that the receiving node can verify blocks before having received the complete message. In order to protect from attacks replacing fragments from a different message with the same block number between same endpoints and same resource at roughly the same time, the MAC 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 which 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.

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. Sid Registration

IANA is requested to enter a new parameter entitled "sid" to the registry "COSE Header Parameters". The parameter is defined in Table 1.

9.2. CoAP Option Number Registration

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

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

9.3. Media Type Registrations

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

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 [[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.

11. References

11.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-20 (work in progress), October 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>.

11.2. Informative References

[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.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.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-02 (work in progress), June 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-04 (work in progress), August 2016.

[I-D.seitz-ace-oscoap-profile]
Seitz, L., "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-00 (work in progress), July 2016.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-02 (work in progress), July 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

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

Let's analyse the contributions one at a time:

- The header parameters of the COSE object are the Context Identifier (Cid) and the Sequence Number (Seq) (also known as 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 depends on the number of simultaneous clients, as discussed in Section 3.2
  - The size of Seq is variable, and increases with the number of messages exchanged.
  - As the IV is generated from the padded Sequence Number and a previously agreed upon context IV it is not required to send the whole IV 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 cipher text 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 cipher text 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 9: 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 extreme 4-byte Cid, based on the assumption of an ACE deployment with billions of clients requesting access to this particular server. (A typical Cid, will be 1-2 byte as is discussed in Appendix A.2.)

- Cid: 0xa1534e3c

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.)

o Seq: 225

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

o Tag is 8 bytes

The COSE object is represented in Figure 10 using CBOR's diagnostic notation.

```
[
  h'a20444a1534e3c0641e2', # protected:
                             {04:h'a1534e3c',
                              06:h'e2'}
  {},                      # unprotected: -
  Tag                      # cipher text + 8 byte authentication tag
]
```

Figure 10: 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 22 bytes, which is the message expansion in this example. The COSE overhead in this example is 22 - (4 + 1 + 8) = 9 bytes, according to the formula in Figure 9. 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 11 summarizes these results.

Tid	Tag	COSE OH	Message OH
5 bytes	8 bytes	9 bytes	22 bytes

Figure 11: Message overhead for a 5-byte Tid 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.

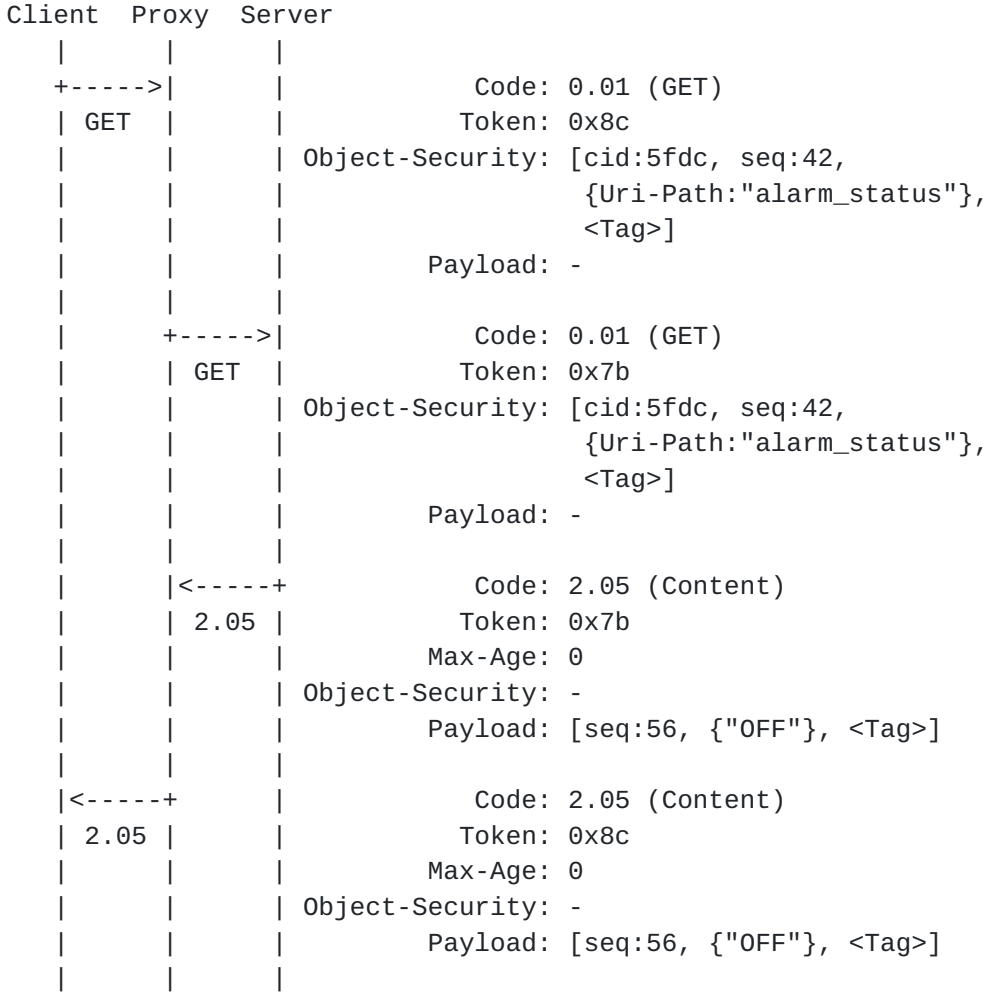


Figure 12: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

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.

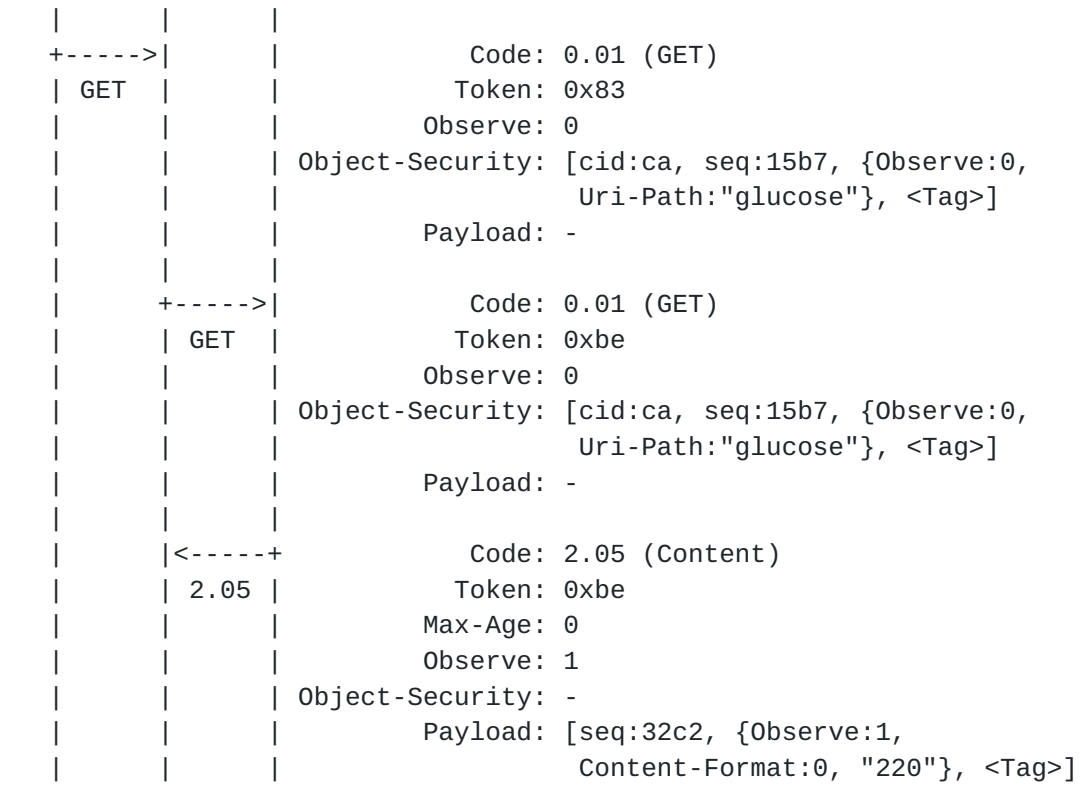
The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

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).



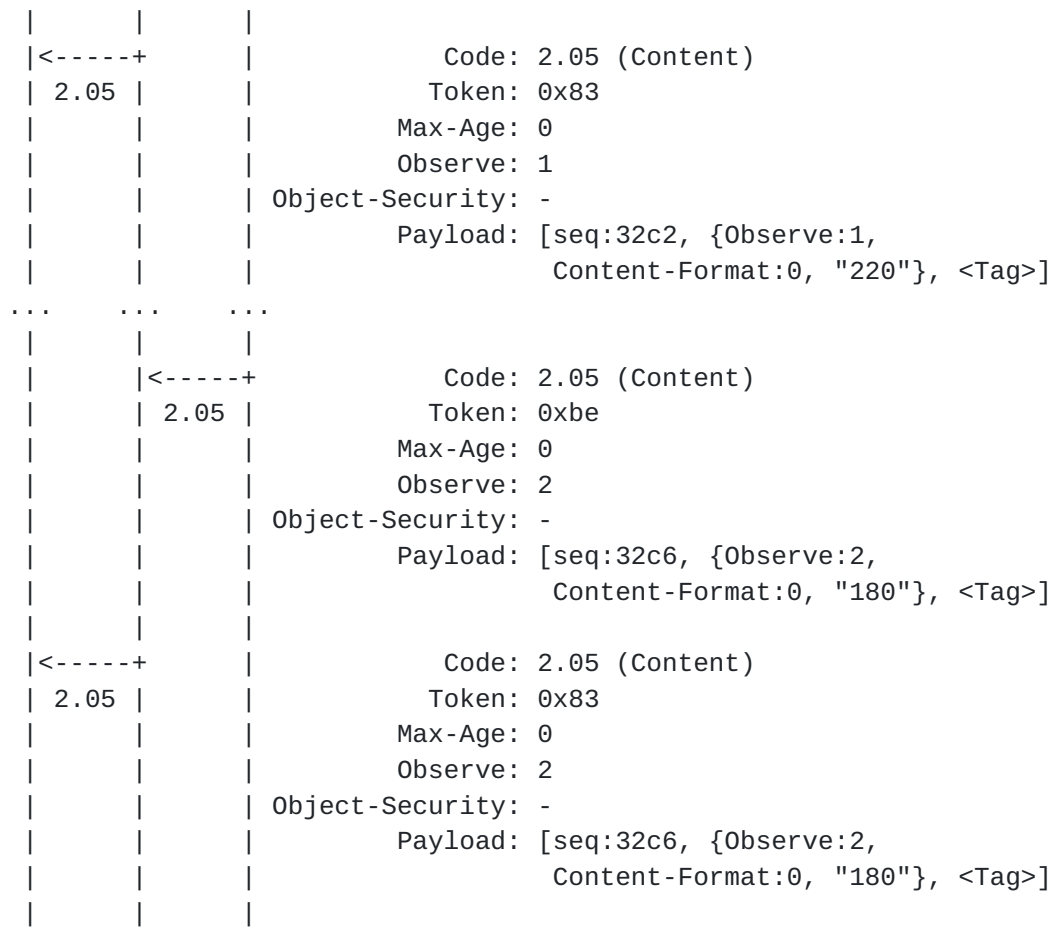


Figure 13: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

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 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 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]).

In the case of symmetric encryption, the same key and IV shall not be used twice. 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 ciphersuites 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 size of the COSE message 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 9.

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 14 summarizes these results.

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

Figure 14: 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 15 summarizes these results.

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

Figure 15: 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 MAC truncated to 8 bytes.

It is assumed that the IV is generated from the Sequence Number and some previously agreed upon context IV. This means it is not required to explicitly send the whole IV in the message.

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                      # cipher text + truncated 8-byte TAG
  ]
)
```

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

Figure 16 summarizes these results.

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

Figure 16: 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                      # cipher text + truncated 8-byte TAG
    ]
  )
```

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

Figure 17 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 17: 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