ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

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

March 21, 2016

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-04

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 Encoded Message Syntax. 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.

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, to improve scalability, efficiency, and uses. 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]; OSCOAP targets the requirements in Sections 3.1 and 3.2.

OSCOAP builds on the CBOR Encoded Message Syntax (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.

OSCOAP 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).

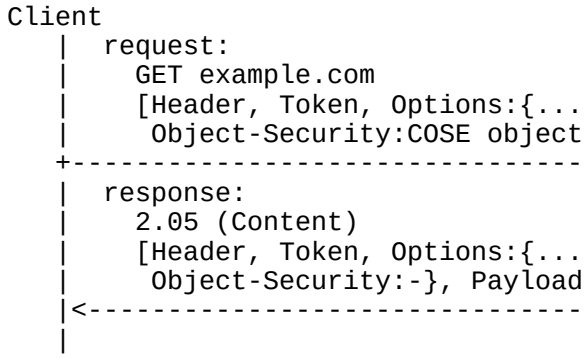


Figure 1: Sketch of OSCOAP

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.

OSCOAP may be used in 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 to payload only of individual messages. We call this object security of content (OSCON) and it is defined in Appendix C. OSCON targets the requirements in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs].

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.

OSCON is 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 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 can 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.

- 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 example of option length is given in Appendix A.

3. The Security Context

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. A security context needs to be pre-established and agreed upon between client and server. How this is done is out of scope of this memo, an example is given in the appendices of [I-D.selander-ace-cose-ecdhe]. Each security context is identified by a Context Identifier, which is unique within a given server. A Context Identifier that is no longer in use can be reassigned to a new security context.

The security context has a "Client Write" part and a "Server Write" part. The client initiating a transaction uses the Client Write part of the context to protect the request; the server receiving the request first uses the Client Write part of the context to verify the request, then the Server Write part of the context to protect the response. Finally, the client uses the Server Write part of the context to verify the response.

OSCOAP is very similar to TLS and borrows mechanisms such as key derivation, and nonce construction from [I-D.ietf-tls-tls13]. The main differences is that OSCOAP uses COSE [I-D.ietf-cose-msg] instead of the TLS record layer, which allows OSCOAP to use a context identifier, and sequence numbers of variable length.

It should be noted that how the context is retrieved within the client and server is linked to the resource discovery, may be implementation specific, and is out of scope of this memo.

An example is shown in Figure 3.

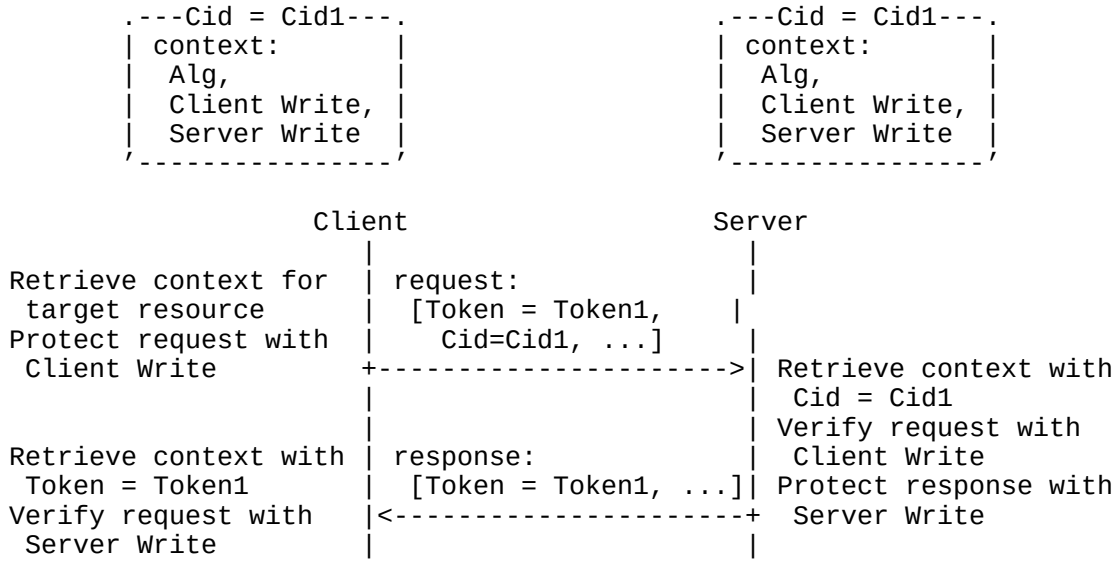


Figure 3: Retrieval and use of the Security Context

The security context structure contains the following parameters:

- Context Identifier (Cid). Variable length byte string that identifies the security context. Immutable.

- Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Immutable.

- Client Write Key. Byte string containing the symmetric key to use in client-sent messages. Length is determined by Algorithm. Immutable.

- Client Write IV. Byte string containing the static IV to use in cryptographic operations on client-sent messages. Length is determined by Algorithm. Immutable.

- Client Write Sequence Number. Non-negative integer enumerating the COSE objects that the client sent, associated to the Context Identifier. It is used for replay protection, and to generate unique nonces. Initiated to 0. Maximum value is determined by Algorithm.

- Server Write Key. Byte string containing the symmetric key to use in server-sent messages. Length is determined by the Algorithm. Immutable.

- Server Write IV. Byte string containing the static IV to use in cryptographic operations on server-sent messages. Length is determined by Algorithm. Immutable.

- Server Write Sequence Number. Non-negative integer enumerating the COSE objects that the server sent, associated to the Context Identifier. It is used for replay protection, and to generate unique nonces. Initiated to 0. 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 default window size is 64.

The size of Cid depends on the number of simultaneous clients, and must be chosen so that the server can uniquely identify the requesting client. Cids of different lengths can be used by different client. In the case of an ACE-based authentication and authorization model [I-D.ietf-ace-oauth-authz], the Authorization Server can define the context identifier of all clients, interacting with a particular server, in which case the size of Cid can be proportional to the logarithm of the number of authorized clients. It is RECOMMENDED to start assigning Cids of length 1 byte (0x00, 0x01, ..., 0xff), and then when all 1 byte Cids are in use, start handling out Cids with a length of two bytes (0x0000, 0x0001, ..., 0xffff), and so on.

The ordered pair (Cid, Client Write 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, and in part of the Additional Authenticated Data (AAD, see Section 5) of the protected CoAP response message.

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. As a consequence, if Proxy-Uri is used, those parts of the URI SHALL be removed from the Proxy-Uri. The CoAP Options Uri-Host, Uri-Port, Proxy-Uri, and Proxy-Scheme SHALL neither be encrypted, nor integrity protected (cf. protection of request URI in Section 5.2).

- The other CoAP options listed in Figure 4 SHALL be encrypted and integrity protected.

No.	C	U	N	R	Name	Format	Length	E	I	D
1	x			x	If-Match	opaque	0-8	x	x	
3	x	x	-		Uri-Host	string	1-255			
4				x	ETag	opaque	1-8	x	x	
5	x				If-None-Match	empty	0	x	x	
6		x	-		Observe	uint	0-3	x	x	x
7	x	x	-		Uri-Port	uint	0-2			
8				x	Location-Path	string	0-255	x	x	
11	x	x	-	x	Uri-Path	string	0-255	x	x	
12					Content-Format	uint	0-2	x	x	
14		x	-		Max-Age	uint	0-4	x	x	x
15	x	x	-	x	Uri-Query	string	0-255	x	x	
17	x				Accept	uint	0-2	x	x	
20				x	Location-Query	string	0-255	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, I=Integrity Protect, D=Duplicate.

Figure 4: Protected CoAP Options

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

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New COAP options SHOULD be encrypted and integrity protected. New COAP options SHALL be integrity protected unless a proxy needs to change the option, and SHALL be encrypted unless a proxy needs to read the option.

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 present 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 "unecrypted" 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 proxies.

- The Observe option is duplicate. If used, then the encrypted Observe and the unencrypted Observe SHALL have the same value. The Observe option as used here targets the requirements of Section 3.2 of [I-D.hartke-core-e2e-security-reqs].

Specifications of new CoAP options SHOULD specify if the option is duplicate and how it are processed with OSCOAP. New COAP options SHOULD NOT be duplicate.

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_Encrypted 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 Client Write Key and the Server Write Key SHALL be 128 bits, the length of the nonce, Client Write IV, and the Server Write IV SHALL be 7 bytes, and the maximum Client Write Sequence Number and Server Write Sequence Number SHALL be 2^56-1. The nonce is constructed exactly like in Section 5.2.2 of [I-D.ietf-tls-tls13], i.e. by padding the Client Write Sequence Number or the Server Write Sequence Number with zeroes and XORing it with the static Client Write IV or Server Write 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_Encrypted structure (Section 2. of [I-D.ietf-cose-msg]). If the COSE object has a different structure, the receiver 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_Encrypted 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 Initialization Vector" parameter. The value is set to the Client Write Sequence Number, or the Server Write Sequence Number, depending on whether the client or server is sending the message. The Partial IV is a byte string (type: bstr), where the length is the minimum length needed to encode the sequence number.
    - If the message is a CoAP request, the "kid" parameter. The value is set to the Context Identifier (see Section 3).
  - The "unprotected" field, which SHALL be empty.
- The "ciphertext" field is computed from the Plaintext and the Additional Authenticated Data (AAD) 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).

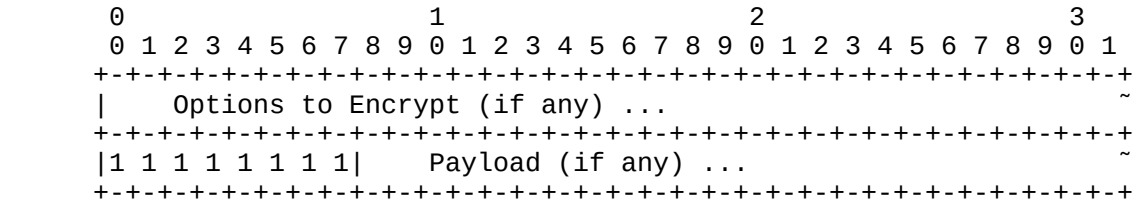


Figure 5: Plaintext

5.2. Additional Authenticated Data

The Additional Authenticated Data ("Enc_structure") as described is Section 5.3 of [I-D.ietf-cose-msg] includes (see Figure 6):

- the "context" parameter, which has value "Encrypted"
- the "protected" parameter, which includes the "protected" part of the "Headers" field;
- the "external_aad" includes:
  - the two first bytes of the CoAP header in the unprotected message (including Version and Code) with Type and Token Length bits set to 0;
  - The Algorithm from the security context used for the exchange;
  - the plaintext request URI composed from the request scheme and Uri-* options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request; and
  - the Transaction Identifier (Tid) of the associated CoAP request, if the message is a CoAP response (see Section 3).

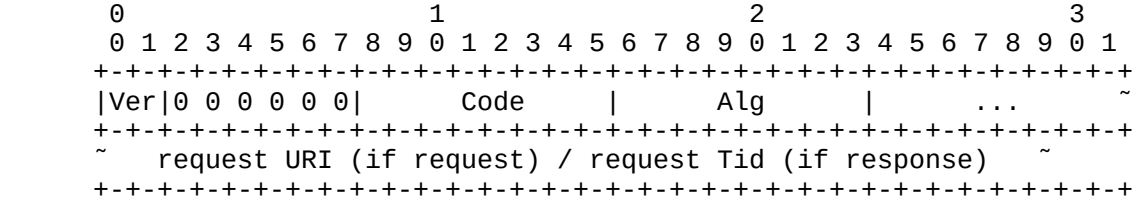


Figure 6: Additional Authenticated Data

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 Client Write Sequence Number, and a Server Write 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. A client uses the Client Write Sequence Number for protecting sent messages and the Server Write Sequence Number for verifying received messages, and vice versa for the server, 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.

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. Note that for the server, the relevant Sequence Number here is the Client Write Sequence Number and vice versa for the client.

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:

1. Increment the Client Write Sequence Number by one (note that this means that sequence number 0 is never used). If the Client Write 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 before this happens. The latter is out of scope of this memo.

2. Compute the COSE object as specified in Section 5

   * the nonce in the AEAD is created by XORing the static IV (Client Write IV) with the partial IV (Client Write Sequence Number).

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 unprotected CoAP message has no Payload, then the value of the Object-Security option is the COSE object. If the unprotected CoAP message has Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

The Client SHALL be able to find the correct security context 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.

3. Compose the nonce by XORing the static IV (Client Write IV) with the Partial IV parameter, received in the COSE Object.

4. Retrieve the Client Write 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 Client Write 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 Server Write Sequence Number by one (note that this means that sequence number 0 is never used). If the Server Write 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 before this happens. The latter is out of scope of this memo.

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

   * The nonce in the AEAD is created by XORing the static IV (Server Write IV) and the Server Write Sequence Number.

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 unprotected CoAP message has no Payload, then the value of the Object-Security option is the COSE object. If the unprotected CoAP message has 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.

3. Compose the nonce by XORing the static IV (Server Write IV) with the Partial IV parameter, received in the COSE Object.

4. Retrieve the Server Write 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 Client Write 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.

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 specification in this memo assumes that there is an established security context. [I-D.ietf-ace-oauth-authz] presents a method for a trusted third party (Authorization Server) to enable key establishment between potentially constrained nodes, using OAuth and PoP Tokens. [I-D.selander-ace-cose-ecdhe] describes a Diffie-Hellman key exchange, authenticated with pre-established keys, and a key derivation method for producing a security context, suitable for OSCOAP. The two methods can be combined, enabling a client and server with relation to a trusted third party to establish a security context with forward secrecy.

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

As the receiver accepts any sequence number larger than the one previously received, the problem of sequence number synchronization is avoided. The alternatives have 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.

Block-wise transfers as currently defined in [I-D.ietf-core-block] cannot be protected end-to-end because the payload as well as the Block1/Block2 options may be changed in an unpredictable way by a proxy. Since [I-D.ietf-core-block] allows for any proxy to fragment the payload, an endpoint receiving a message fragment with a block option is not able to verify integrity of that fragment. As a consequence, block-wise disables end-to-end security: an adversary may inject an unlimited number of messages with a block option claiming it to be a sequence of message fragments without the receiving endpoint being able to disprove the claim.

If instead the payload and block options Block1/Block2 were not allowed to be changed by intermediate devices, then the message fragments could be integrity protected end-to-end. In that case each individual block can be securely verified by the receiver, retransmission securely requested etc. Since the blocks are enumerated sequentially, and carry information about whether this fragment is the last, when all blocks have been securely received is enough to prove that the entire message has been securely transferred.

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, provides mitigation against attacks on sensor and actuator communication, which may have a direct impact 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 Number Registration

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

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

9.2. 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

9.3. 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

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

11. References

11.1. Normative References

[I-D.ietf-cose-msg] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-10 (work in progress), February 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>.

11.2. Informative References

[I-D.hartke-core-e2e-security-reqs] Selander, G., Palombini, F., Hartke, K., and L. Seitz, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-00 (work in progress), March 2016.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0", draft-ietf-ace-oauth-authz-01 (work in progress), February 2016.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 2015.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-00 (work in progress), March 2016.

[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 unprotected CoAP message has 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 has 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 Ciphertext (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, and must be chosen so that the server can uniquely identify the requesting client. For example, in the case of an ACE-based authentication and authorization model [I-D.ietf-ace-oauth-authz], the Authorization Server or the server itself can define the context identifier of all clients interacting with a particular server, in which case the size of Cid can be proportional to the logarithm of number of authorized clients.
    - As Cids of different lengths can be used by different client, it is RECOMMENDED to start assigning Cids of length 1 byte (0x00, 0x01, ..., 0xff), and then when all 1 byte Cids are in use, start handling out Cids with a length of two bytes (0x0000, 0x0001, ..., 0xffff).
  - The size of Seq is variable, and increases with the number of messages exchanged.
  - As the nonce is generated from the padded Sequence Number and a previously agreed upon static IV it is not required to send the whole nonce in the message.
- The Ciphertext, 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 the OSCOAP mandatory 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 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.)

- 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'a20444a1534e3c0641e2', # protected:
                              {04:h'a1534e3c',
                               06:h'e2'}
   {},                      # unprotected: -
   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 Ciphertext 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 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.

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

Figure 9: 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 Actuator

Here is an example targeting the scenario in Section 3.1 of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting valve 34 to be turned to position 3 (PUT /valve34 with payload value "3"), and getting a confirmation. The CoAP options Uri-Path and Payload are encrypted and integrity protected,

and the CoAP header field Code is integrity protected (see Section 4).

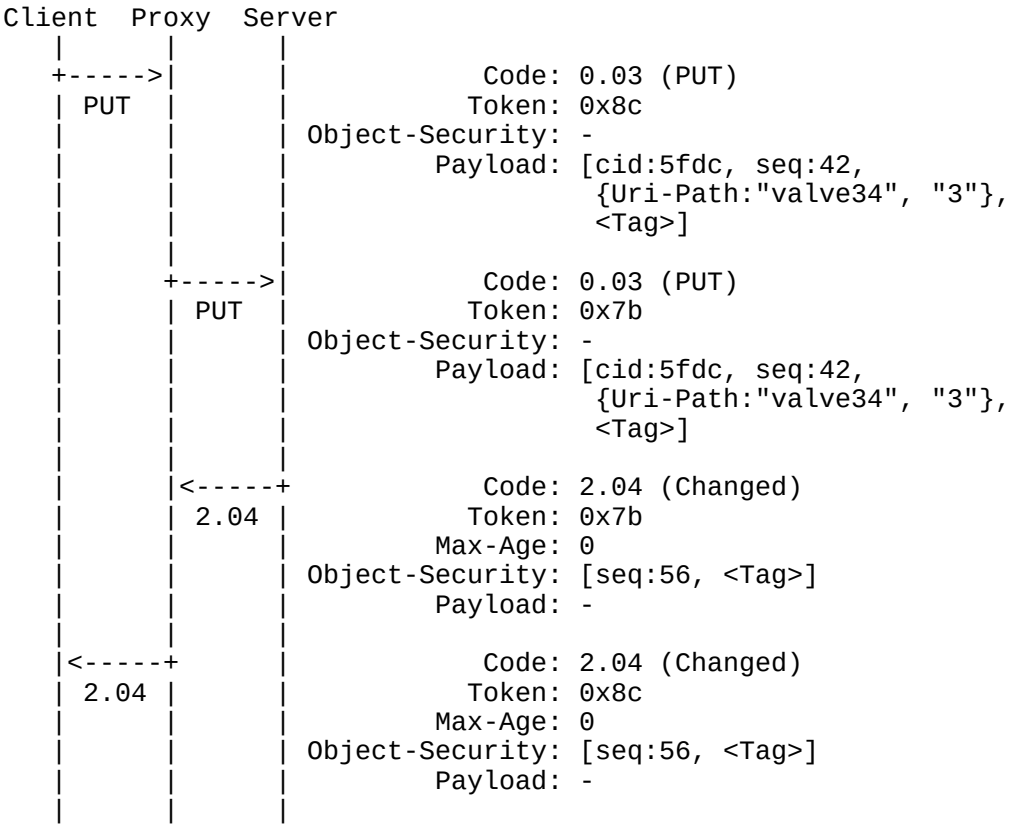


Figure 10: Indication of CoAP PUT protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

Since the unprotected request message (PUT) has payload ("3"), the COSE object (indicated with [...]) is carried as the CoAP payload. Since the unprotected response message (Changed) has no payload, the Object-Security option carries the COSE object as its value.

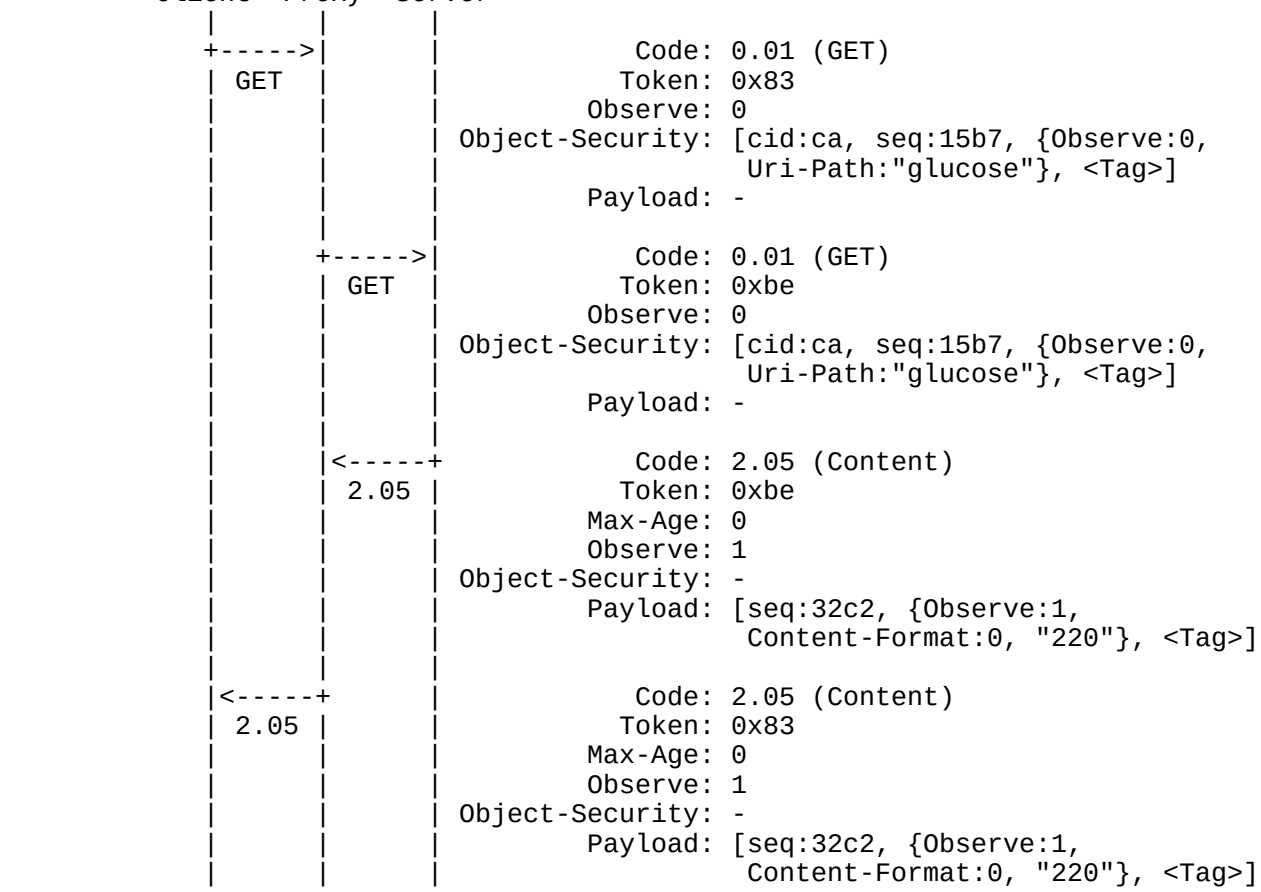
The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

The option Uri-Path (valve34) and payload ("3") 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 sent request message (see Section 6.1).

B.2. Secure Subscribe to Sensor

Here is an example targeting the scenario in Section 3.2 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).



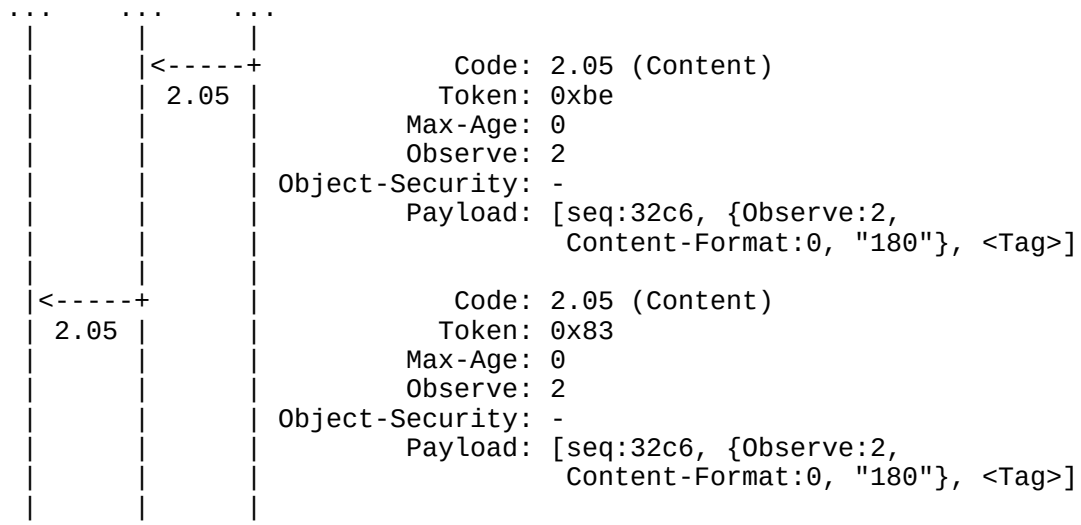


Figure 11: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

Since the unprotected request message (GET) has 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 in Section 3.1 and 3.2 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 Sections 3.3 - 3.5 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 nonce SHALL NOT be used twice. The use of sequence numbers for partial IV as specified for OSCOAP MAY be used. of sequence numbers for replay protection as described in Section 6.1 MAY be used. 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 receivers 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 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 MAC truncated to 8 bytes.

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

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

The object in COSE encoding gives:

```
993(                          # COSE_Encrypted_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_Encrypted_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' that is used here to sign a COSE_Encrypted_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
  993(                          # COSE_Encrypted_Tagged
    [
      h'a20444a1534e3c0641a3', # protected:
                                 {04:h'a1534e3c',
                                  06:h'a3'}
      {7:SIG},                 # unprotected:
                                  07: 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_Encrypted_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