ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

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

October 19, 2015

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-03

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options and header fields, and a secure binding between CoAP request and response messages. Using this method the unprotected CoAP message is transformed into a protected

CoAP message, which contains a secure data object protecting the unprotected message, and which is sent instead of the unprotected message. The use of OSCOAP is signaled with the Object-Security option, also defined in this memo.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].These words may also appear in this document in lowercase, absent their normative meanings.

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

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

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

Terminology for authentication and authorization in constrained environments, such as "Authorization Server", "Resource Server", etc, is defined in [I-D.ietf-ace-actors].

The CoAP option Object-Security and the Secure Message (SM) format are defined in this memo.

Two different scopes of object security are defined:

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

o OSCON = object security of content, signaled with Content Format/ Media Type set to application/oscon.

OSCON is defined in Appendix C and included for comparison with OSCOAP.

The COSE message format is defined in [I-D.ietf-cose-msg].

2. Background

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

For constrained-node networks there may be several reasons for messages to be cached or stored in one node and later forwarded.

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

Based on these examples the following security requirements have been identified:

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

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

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

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

5. The CoAP header fields "Code" and "Version" shall be integrity protected between Client and Server.

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

In this list above, requirements 1-2 deals essentially with protecting the CoAP payload only, whereas 3-6 deals with protecting an entire CoAP request-response exchange, including also CoAP options and header fields.

Object Security of CoAP (OSCOAP), which is the main focus of this memo, addresses all requirements above by defining a method for

encryption, integrity protection and replay protection of CoAP payload, options and header fields, and a secure binding between CoAP request and response messages. OSCOAP consists of:

o the Object-Security option, indicating that OSCOAP is being used;

o a compact cryptographic message format called "Secure Message", based on the COSE message format ([I-D.ietf-cose-msg]); and

o a scheme for transforming an unprotected CoAP message into a protected CoAP message, which contains the Object-Security option and a Secure Message protecting CoAP payload, options and header fields.

The same method can be applied to payload only of individual messages, targeting only requirements 1-2 above. We call this object security of content (OSCON) and it is defined in Appendix C.

Examples of the use of OSCOAP and OSCON are given in Appendix D.

3. The Object-Security Option

In order to end-to-end protect CoAP message exchanges including options and headers, a new CoAP option is introduced: the Object-Security option. The Object-Security option indicates that OSCOAP is used, i.e. that certain CoAP Header fields, Options and Payload (if present) are integrity and replay protected and potentially encrypted, using a cryptographic message format called the Secure Message format Section 4.

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

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

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

Figure 1: The Object-Security Option

The length of the option depends on the specific choice of the Secure Message format. Length 0 indicates that the Secure Message is the CoAP Payload of the message, and is used when the CoAP message type used supports payload.

4. Secure Message Format

There exist already standardized and draft content formats for encryption and integrity protection of data such as CMS [RFC5652], JWS [RFC7515], JWE [RFC7516], and COSE [I-D.ietf-cose-msg].

Current CMS and JWx objects are undesirably large for very constrained devices. Large messages has a negative impact on memory and storage in constrained devices, packet fragmentation in constrained-node networks due to limited frame sizes, and increased energy consumption due to more data transmission and reception. The candidate for use with object security of CoAP messages is the COSE message format [I-D.ietf-cose-msg].

Pending an optimized and stable version of the COSE message format this memo defines the SM format to refer to a content format for encrypted and integrity protected data, and also includes a unique transaction identifier for replay protection. Appendix A shows a profile of the COSE message format which complies with the Secure Message format.

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

4.1. Secure Message Header

The following parameters SHALL be included in the SM Header:

o Context Identifier (CID). This parameter identifies the sender security context including the cipher suite, key(s) and additional algorithm specific parameters used to protect the message. Each client and server communicating using OSCOAP has two contexts, one for sending and one for receiving.

o Sequence Number (SEQ). The Sequence Number parameter enumerates the Secure Messages sent associated to a Context Identifier, and is used for replay protection and uniqueness of nonce. The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The granularity of "sender" - what is being identified with the Context Identifier - is defined by the application. With OSCOAP the Context Identifier typically identifies the sending party and different resources may be identified by the Uri-Path in the request. (Compare Appendix C.)

The ordered sequence (SEQ, CID) is called Transaction Identifier (TID), and SHALL be unique for each SM.

4.2. Secure Message Body and Tag

The use cases require support for two message types, one for Encryption and Integrity Protection, and another for integrity protection only. The SM Body and the SM Tag are different depending on message type.

For Integrity Protection Only we denote by Authenticated Data (AD) the data which is integrity protected in the Secure Message. For Encryption and Integrity Protection we denote by Plaintext and Additional Authenticated Data (AAD), the data which is encrypted and integrity protected, and integrity protected only, respectively, in the Secure Message.

The message type SHALL be explicit to allow an intermediate node to distinguish between the two types and read the SM Body of an Integrity Protected Only message.

4.2.1. Integrity Protection Only

In the case of integrity protection only, the SM Body SHALL consist of the payload of the CoAP message.

The SM Tag SHALL consist of the Signature / Message Authentication Code (MAC) as defined by the cipher suite calculated over the Authenticated Data (AD). The AD for OSCOAP is defined in Section 5.1.2.

4.2.2. Encryption and Integrity Protection

The use cases require support for two kinds of cipher suites: Authenticated Encryption with Additional Data (AEAD) as well as Symmetric Encryption and Asymmetric Signature (SEAS).

In case of AEAD, the SM Body and SM Tag SHALL consist of the Ciphertext as defined by the cipher suite calculated over the Plaintext and the Additional Authenticated Data (AAD).

In case of SEAS, the SM Body SHALL be the Ciphertext as defined by the symmetric encryption algorithm, given by the cipher suite, calculated over the Plaintext. The SM Tag SHALL be the Signature defined by the cipher suite calculated over Ciphertext and AAD.

The Plaintext and the AAD for OSCOAP are defined in Section 5.2.2.

5. CoAP Message Protection

This section presents how OSCOAP protects individual CoAP messages including payload, options and header fields, as well as request-response message exchanges, using the Object-Security option (Section 3) and the Secure Message format (Section 4).

The basic idea is that the significant parts of an unprotected CoAP message - including payload, certain header field and options - are protected using the Secure Message format and sent in a CoAP message with the Object-Security option, in what we then call a "protected" CoAP message. As much a possible of the CoAP message should be protected, but not all CoAP header fields or options can be encrypted and integrity protected, because some are intended to be read or changed by an intermediary node, see Section 6.1 and Section 6.2.

The use of OSCOAP is signaled with the Object-Security option. Endpoints supporting the Object-Security option MUST verify the SM as described in this section before accepting a message as valid. An endpoint receiving a CoAP request with the Object-Security option MUST respond with a CoAP message with the Object-Security option.

The differences between Encryption and Integrity Protection vs Integrity Protection Only is described below. Encryption and Integrity Protection SHALL be used by default.

5.1. Integrity Protection Only

5.1.1. Protected CoAP message formatting

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

- o The CoAP header SHALL be the same as the unprotected CoAP message.
- o The CoAP options SHALL consist of the same options as the unprotected CoAP message, and the Object-Security option.
- o If the unprotected CoAP message has no Payload then the Object-Security option SHALL contain the SM. If the unprotected CoAP message has Payload, then the Object-Security option SHALL be empty and the Payload of the CoAP message SHALL be the SM.

5.1.2. Secure Message formatting

The SM Header, Body and Tag are specified in Section 4.1 and Section 4.2.

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

- the SM Header;
- the two first bytes of the CoAP header (including Version and Code) with Type and Token Length bits set to 0;
- all CoAP options present which are marked as IP in Figure 2 (Section 6.2), in the order as given by the option number (each Option with Option Header including delta to previous IP-marked Option which is present);
- the CoAP Payload (if any); and
- the Transaction Identifier of the associated CoAP Request, if the message is a CoAP Response (see Section 4.1).

5.1.3. Integrity Protection and Verification

A CoAP endpoint protecting a CoAP message with the Object-Security option using a cipher suite for integrity protection only SHALL generate a protected CoAP message and SM based on the unprotected CoAP message as described in Section 5.1.1 and Section 5.1.2. In addition, the sending endpoint SHALL process the Sequence Number as described in Section 7.

A CoAP endpoint receiving a message containing the Object-Security option SHALL first recreate the Authenticated Data as described in Section 5.1.2, and then verify the SM Tag as defined by the cipher suite associated to the Context Identifier. In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 7.

5.2. Encryption and Integrity Protection

5.2.1. Protected CoAP message formatting

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

- The CoAP header SHALL be the same as the unprotected CoAP message.

- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message (those not marked as E in Figure 2 (Section 6.2)), and the Object-Security option. The options shall be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).

- If the unprotected CoAP message has no Payload then the Object-Security option SHALL contain the SM. If the unprotected CoAP message has Payload, then the Object-Security option SHALL be empty and the Payload of the CoAP message SHALL be the SM.

5.2.2. Secure Message formatting

The SM Header, Body and Tag are specified in Section 4.1 and Section 4.2.

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

- the SM Header;

- the two first bytes of the CoAP header (including Version and Code) with Type and Token Length bits set to 0;

- all CoAP options present which are marked as IP but not marked as E in Figure 2 (Section 6.2), in the order as given by the option number (each Option with Option Header including delta to previous IP-marked Option which is present); and

- the Transaction Identifier of the associated CoAP Request, if the message is a CoAP Response (see Section 4.1).

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

- all CoAP Options present which are marked as E in Figure 2 (see Section 6.2), in the order as given by the Option number (each Option with Option Header including delta to previous E-marked Option); and

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

5.2.2.1. Encryption and Decryption

A CoAP endpoint protecting a CoAP message with the Object-Security option using a cipher suite for encryption and integrity protection SHALL generate a protected CoAP message and SM based on the

unprotected CoAP message as described in Section 5.2.1 and Section 5.2.2. In addition, the sending endpoint SHALL process the Sequence Number as described in Section 7.

A CoAP endpoint receiving a message containing the Object-Security option SHALL recreate the Additional Authenticated Data as described in Section 5.1.2 and verify the integrity of, and decrypt the message as defined by the cipher suite associated to the Context Identifier. In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 7.

6. Protected CoAP Message Fields

The CoAP payload SHALL be integrity protected. The CoAP payload SHOULD be encrypted by default.

How CoAP Options and Header Fields shall be protected is described in the remainder of this section.

6.1. Protected CoAP Header Fields

This section describes which CoAP header fields are encrypted or integrity protected end-to-end in OSCOAP.

The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length may be changed by a proxy and thus SHALL neither be integrity protected nor encrypted.

The Version and Code fields SHALL be integrity protected, see security considerations.

6.2. Protected CoAP Options

This section describes which CoAP options are encrypted and integrity protected, if present in the unprotected CoAP message.

All CoAP options SHALL be encrypted by default, unless intended to be read by an intermediate node; and SHALL be integrity protected, unless intended to be changed by an intermediate node.

However, some special considerations are necessary because CoAP defines certain legitimate proxy operations, because the security information itself may be transported as an option, and because different processing is performed depending on whether encryption is applied or not.

The details are presented in Section 6.2.1 and Section 6.2.2, and summarized in Figure 2.

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

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

Figure 2: Protected CoAP options in OSCOAP

CoAP options marked "i" indicate that they are used as invariants in the authenticated data (AD/AAD) as described in Section 6.2.1.1 and Section 6.2.1.2.

In case of Integrity Protection Only, options marked with "a" and "b" are composed into a URI as described in Section 6.2.1.2 and included as invariant in the Proxy-Uri option in the Authenticated Data.

In case of Encryption and Integrity Protection, options marked "a" are composed into a URI as described in Section 6.2.2 and included as the Proxy-Uri option in the Additional Authenticated Data. (Options marked "b" are included in the Plaintext.)

6.2.1. Integrity Protection

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

o CoAP options of the unprotected message which are Safe-to-Forward SHALL be integrity protected. See Figure 2.

Note: The Object-Security option in itself is Safe-to-Forward but is added to the protected message.

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

- o Max-Age, Observe, Block1, Block2: These options may be modified by a proxy in a way that is not predictable for client and server.

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

6.2.1.1. Proxy-Scheme

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

- o If there is a Proxy-Scheme present in the unprotected message, then the client SHALL integrity protect the Proxy-Scheme option.

- o If there is no Proxy-Scheme option present the client SHALL include the Proxy-Scheme option in the authenticated data (AD/AAD) set to the URI scheme. (The sent message does not include the Proxy-Scheme option.)

- o The server SHALL insert the Proxy-Scheme option with the name of the URI scheme the message was received in the authenticated data (AD/AAD).

6.2.1.2. Uri-*

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

The Proxy-Uri option is defined in this memo to be an invariant with respect to the following processing (applied to Integrity Protection only, for Encryption see next section):

- o If there is a Proxy-Uri present, then the client MUST integrity protect the Proxy-Uri option and the Uri-* options MUST NOT be integrity protected.

- o If there is no Proxy-Uri option present, then the client SHALL compose the full URI from Uri-* options according to the method described in section 6.5 of [RFC7252]. The Authenticated Data contains the following options, modified compared to what is sent:

- o All Uri-* options removed

- o A Proxy-Uri option with the full URI included

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

6.2.2. Encryption

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

- o Max-Age, Observe, Block1, Block2, Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.

- o Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.

- o Object-Security: This is the security-providing option.

In the case of encryption, the Proxy-Uri of the Additional Authenticated Data MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not necessarily available to a Forward Proxy.

7. Replay Protection and Freshness

In order to protect from replay of messages and verify freshness of responses, a CoAP endpoint using object security SHALL maintain Sequence Numbers (SEQs) of sent and received Secure Messages (see Section 4.1), associated to the respective security context identified with the Context Identifier (CID).

7.1. Replay Protection

An endpoint SHALL maintain a SEQ for each security context it uses to receive messages, and one SEQ for each security context for protecting sent messages. Depending on use case, an endpoint MAY maintain a sliding receive window for Sequence Numbers in received messages associated to each CID, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message a sending endpoint SHALL step the SEQ of the associated CID. However, if the Sequence Number counter wraps, the endpoint must first acquire a new CID and associated security context/key(s). The latter is out of scope of this memo.

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

7.2. Freshness

OSCOAP is a challenge-response protocol, where the response is verified to match a prior request by including the unique transaction identifier TID (concatenation of SEQ and CID) of the request in the integrity calculation of the response message.

If a CoAP server receives a request with the Object-Security option, then the authenticated data (AD or AAD) of the response SHALL include the TID of the request as described in Section 5.1.2 and Section 5.2.2.

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 authenticated data (AD or AAD) as described in Section 5.1.2 and Section 5.2.2.

8. Security Considerations

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

DTLS protects the entire CoAP message including Header, Options and Payload, whereas OSCOAP protects the payload and message fields described in Section 6.1 and Section 6.2. The cost for DTLS providing this protection is the overhead in e.g. additional messages, processing, memory incurred by the DTLS Handshake protocol, which can be omitted in use cases where key establishment can be provided by other means.

CoAP specifies how messages should be acknowledged on message layer. The CoAP message layer, however, cannot be protected by application layer security end-to-end 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 object security solution, since message layer is excluded from that.

The CoAP Header field Code needs to be integrity protected end-to-end. For example, if a malicous man-in-the-middle would replace the client requested GET with a DELETE, this must be detected by the server. The CoAP Header field Version needs also to be integrity protected to prevent from potential cross-version attacks, such as bidding-down.

Blockwise transfers as defined [I-D.ietf-core-block] cannot be protected with application layer security end-to-end because the Block1/Block2 options may be changed in an unpredictable way by an intermediate node.

However, it is possible to define end-to-end block options analogous to Block1 and Block2 which are safe-to-forward, integrity protected and not supposed to be changed by intermediate devices. With such an option each individual block can be securely verified by the receiver, retransmission securely requested etc. Since the blocks are enumerated sequentially and carry information about last block, when all blocks have been securely received, this proves that the entire message has been securely transferred.

The Observe option cannot be integrity protected since it is allowed to change in an unpredictable way. But since message sequence numbers are integrity protected a client
can verifies that a GET response has not been received before.

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

This specification needs to be complemented with a procedure whereby the client and the server establish the keys used for wrapping and unwrapping the Secure Message. One way to address key establishment is to assume that there is a trusted third party which can support

client and server, such as the Authorization Server in [I-D.ietf-ace-actors]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server which can be use to derive the keys used in the Secure Message exchange. Similarly, the Authorization Server may, on behalf of the server, notify the client of server supported ciphers, in order to facilitate the usage of OSCOAP in deployments with multiple supported cryptographic algorithms.

The security contexts required are different for different cipher suites. For an AEAD or SEAS it is required to have a unique Initialization Vector for each message, for which the Sequence Number is used. The Initialization Vector SHALL be the concatenation of a Salt (4 bytes unsigned integer) and the Sequence Number. The Salt SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it in each message. For example, the Salt may be established by the same means as keys are established.

9. Privacy Considerations

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

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

10. IANA Considerations

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

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

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

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

Media Type: application/oscon

Encoding: -

Id: 70

Reference: [this document]

11. 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 new versions of the draft.

12. References

12.1. Normative References

[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>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

12.2. Informative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), September 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-09 (work in progress), October 2015.

[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-cose-msg]
Schaad, J. and B. Campbell, "CBOR Encoded Message Syntax", draft-ietf-cose-msg-05 (work in progress), September 2015.

[I-D.seitz-ace-core-authz]
Seitz, L., Selander, G., and M. Vucinic, "Authorization for Constrained RESTful Environments", draft-seitz-ace-core-authz-00 (work in progress), June 2015.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

Appendix A. COSE Profile of SM

This section defines a profile of the 05-version of COSE [I-D.ietf-cose-msg] complying with the Secure Message format (see Section 4) and supporting the two scopes of object security OSCOAP and OSCON (Appendix C). In the last subsection we elaborate on possible optimizations.

- The "COSE_MSG" top level object as defined in COSE corresponds to the Secure Message object.
- The "msg_type" parameter corresponds to the Secure Message type, as defined in Section 4.2. Depending on the use case, this field can take the values msg_type_mac, msg_type_signed or msg_type_encryptData.
- The "Header" field of the COSE object corresponds to the Header field of the Secure Message.
    - The "protected" field includes:
        - the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).
    - The "unprotected" field is empty.

A.1. Integrity Protection Only

When Integrity Protection only needs to be provided, the Secure Message object corresponds to a COSE_MSG with msg_type equal to msg_type_signed (COSE_Sign) or msg_type_mac (COSE_mac).

The Externally Supplied Data ("external_aad" field), as defined in Section 4.1 of [I-D.ietf-cose-msg] include the Authenticated Data as defined in Section 5.1.2 with the exception of SM Header and CoAP Payload.

A.1.1. COSE_Sign

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

The "Headers" field of COSE_Sign as defined in Appendix A.

The "payload" field contains the CoAP Payload (if any).

The "signatures" array contains one "COSE_signature" item. The "Headers" field of the COSE_signature object is defined as follows:

- The "protected" field includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

- The "unprotected" field is empty.

The "signature" field contains the computed signature value as described in Section 4.2 of [I-D.ietf-cose-msg].

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

A.1.2. COSE_mac

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

The "Headers" field of COSE_mac as defined in Appendix A.

The "payload" field contains the CoAP Payload (if any).

The "tag" field contains the MAC value, computed as defined in Section 6.1 of [I-D.ietf-cose-msg].

The "recipients" array contains one "COSE_recipient" item (section 5 of [I-D.ietf-cose-msg]). The "COSE_recipient" item contains one "COSE_encrypt_fields" object. The "Headers" field of the COSE_encrypt_fields object is defined as follows:

- The "protected" field includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

- The "unprotected" field is empty.

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

A.2. Encryption and Integrity Protection: COSE_enveloped

When Encryption and Integrity Protection need to be provided, the Secure Message object corresponds to a COSE_MSG with msg_type equal to msg_type_enveloped (COSE_enveloped).

The Additional Authenticated Data ("Enc_structure") as described is Section 5.3 of [I-D.ietf-cose-msg] is defined in Section 5.2.2: * the "protected" parameters includes the SM Header; * the "external_aad" includes the other fields (CoAP Version, Code, Options to integrity protect and TID).

The plain text, as mentioned in Sections 5.3 and 5.4 of [I-D.ietf-cose-msg] is defined in Section 5.2.2 and contains CoAP Options to encrypt and the CoAP Payload.

A COSE_MSG of type COSE_enveloped [I-D.ietf-cose-msg] is a Secure Message if its fields are defined as follows (see example in Appendix B.3).

The "Headers" field of COSE_encrypt_fields item as defined in Appendix A.

The "ciphertext" field is encoded as a nil type, following the specifications in Section 5.1 of [I-D.ietf-cose-msg].

The "recipients" array contains one "COSE_recipient" item (Section 5.1 of [I-D.ietf-cose-msg]). The "COSE_recipient" item contains one "COSE_encrypt_fields" object. The "Headers" field of the COSE_encrypt_fields object is defined as follows:

o The "protected" field includes:

   * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

o The "unprotected" field is empty.

The "ciphertext" field of the COSE_encrypt_fields object contains the encrypted plain text, as defined in section 5 of [I-D.ietf-cose-msg].

A.3. COSE Optimizations

For constrained environments it is important that the message expansion due to security overhead is kept at a minimum.

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

1. The first improvement proposed is to flatten the structure of the COSE_msg, following the Encrypted COSE structure defined in

Section 5.2 of [I-D.ietf-cose-msg]. In fact, there is little need to support multiple signatures or recipients in the use cases targeting the most constrained devices. Two different structures inspired by the COSE_encryptData are defined: COSE_ip and COSE_en. COSE_ip is used for the Integrity Protection Only use case (Section 5.1), COSE_en is used for Encryption (Section 5.2).

2. In general, the security context defines uniquely the cipher suite, and hence the "alg" parameter of COSE_msg can be removed.

3. The "unprotected" field is not used since it is assumed that all parameters should be protected when possible. Thus the "Headers" structure can be flattened into a "protectedHeader" field, containing the "cid" parameter and the "seq" parameter.

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

5. Digitally signed messages have the largest absolute overhead due to the size of the signature (see Appendix B.2 and Appendix B.4). Whereas certain MACs can be securely truncated, signatures cannot. Signature schemes with message recovery allow some remedy since they allow part of the message to be recovered from the signature itself and thus need not be sent. The effective size of the signature could in this way be considerably reduced, which would have a large impact on the message size (compare size of signature and total overhead in Figure 5 and Figure 6). A valuable optimization would thus be to support signature schemes with message recovery.

Combining the first 4 points, the resulting structures and their fields are defined as follows: COSE_ip top level object corresponds to the Secure Message object.

- The "msg_type" parameter takes a new value, msg_type_integrityprotection=5.

- The "protectedHeader" field, analogous to the "protected" field of the "Headers", includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

  * the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).

- The "payload" field (as described in Appendix A.1.1 and Appendix A.1.2).

- The "tag" field (as described in Appendix A.1.1 and Appendix A.1.2).

COSE_en top level object corresponds to the Secure Message object.

- The "msg_type" parameter takes a new value, msg_type_encryption=6.

- The "protectedHeader" field, analogous to the "protected" field of the "Headers", includes:

  * the new "cid" parameter which corresponds to the parameter Context Identifier of the Secure Message (see Section 4.1);

  * the new "seq" parameter corresponding to the parameter Sequence Number of the Secure Message (see Section 4.1).

- The "ciphertext" field (as described in Appendix A.2).

- The "tag" field contains the tag value in case Integrity Protection is also provided.

Appendix B. Comparison of message sizes

This section gives some examples of overhead incurred with the current proposal for COSE at the time of writing [I-D.ietf-cose-msg]. Message sizes are also listed for a modified version of COSE implementing some of the optimizations described in Appendix A.3 and for a lower bound CBOR encoding of the Secure Message with structure [seq, cid, body, tag].

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

The size of the header is shown separately from the size of the MAC/signature. An 8-byte Context Identifier and a 3-byte Sequence Number are used throughout all examples, with these value:

- cid: 0xa1534e3c5fdc09bd

- seq: 0x112233

For each scheme, we indicate the fixed length of these two parameters ("seq+cid" column) and of the tag ("MAC"/"SIG"/"TAG"). The "Total Size" column shows the total Secure Message size, while the "Overhead" column is calculated from the previous columns following this equation:

Overhead = Total Size - (MAC + seq+cid)

This means that overhead incurring from CBOR encoding is also included in the Overhead count.

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

B.1. MAC Only

This example is based on HMAC-SHA256, with truncation to 16 bytes.

The object in COSE encoding gives:

```
[
  3,                                        # msg_type
  h'a20104637365714311223',                 # protected:
                                              {1: 4,
                                              "seq": h'112233'}
  {},                                       # unprotected
  h'',                                      # payload
  MAC,                                      # truncated 16-byte MAC
  [                                         # recipients
    [                                       # recipient structure
      h'',                                  # protected
      {1:-6, "cid":h'a1534e3c5fdc09bd'},    # unprotected
      h''                                   # ciphertext
    ]
  ]
]
```

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

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  5,                                        # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',      # protected:
                                              {11:h'a1534e3c5fdc09bd',
                                               12:h'112233'}
  h'',                                      # payload
  MAC                                       # truncated 16-byte MAC
]
```

This modified COSE object encodes to a total size of 37 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                # seq
  h'a1534e3c5fdc09bd',                      # cid
  h'',                                      # payload
  MAC                                       # truncated 16-byte MAC
]
```

This object encodes to a total size of 32 bytes.

Figure 3 summarizes these results.

Scheme	seq+cid	MAC	Total Size	Overhead
COSE	11 B	16 B	53 bytes	26 bytes
mod-COSE	11 B	16 B	37 bytes	10 bytes
bound	11 B	16 B	32 bytes	5 bytes

Figure 3: Comparison of COSE, modified COSE and CBOR lower bound for HMAC-SHA256.

B.2. Signature Only

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

The object in COSE encoding gives:

```
[
  1,                                        # msg_type
  h'a16373657143112233',                    # protected:
                                              {"seq": h'112233'}
  {},                                       # unprotected
  h'',                                      # payload
  [                                         # signatures
    [                                       # signature structure
      h'a201266363696448a1534e3c5fdc09bd', # protected:
                                              {1: -7,
                                            "cid":h'a1534e3c5fdc09bd'}
      {},                                   # unprotected
      SIG                                   # 64-byte signature
    ]
  ]
]
```

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

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
[
  5,                                        # msg_type
  h'a20b48a1534e3c5fdc09bd0c43112233',      # protected:
                                              {11:h'a1534e3c5fdc09bd',
                                               12:h'112233'}
  h'',                                      # payload
  SIG                                       # 64-byte signature
]
```

The COSE object encodes to a total size of 86 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
[
  h'112233',                                # seq
  h'a1534e3c5fdc09bd',                      # cid
  h'',                                      # payload
  SIG                                       # 64-byte signature
]
```

This object encodes to a total size of 81 bytes.

Figure 4 summarizes these results.

Scheme	seq+cid	SIG	Total Size	Overhead
COSE	11 B	64 B	100 bytes	25 bytes
mod-COSE	11 B	64 B	86 bytes	11 bytes
bound	11 B	64 B	81 bytes	6 bytes

Figure 4: Comparison of COSE, modified COSE and CBOR lower bound for 64 byte ECDSA signature.

B.3. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-128-CCM-8.

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

The object in COSE encoding gives:

```
[
  2,                                        # msg_type
  h'a20104637365714311223',                # protected:
                                              {1: 4,
                                              "seq": h'112233'}
  {},                                       # unprotected
  TAG,                                      # 8byte authentication tag
  [                                         # recipients
    [                                       # recipient structure
      h'',                                  # protected
      {1:-6, "cid":h'a1534e3c5fdc09bd'},    # unprotected
      h''                                   # ciphertext
    ]
  ]
]
```

The COSE object encodes to a total size of 44 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
   [
     6,                                         # msg_type
     h'a20b48a1534e3c5fdc09bd0c43112233',       # protected:
                                                  {11:h'a1534e3c5fdc09bd',
                                                   12:h'112233'}
     h'',                                       # ciphertext
     TAG                                        # 8byte authentication tag
   ]
```

The modified COSE object encodes to a total size of 29 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
   [
     h'112233',                                 # seq
     h'a1534e3c5fdc09bd',                       # cid
     h'',                                       # ciphertext
     TAG                                        # 8byte authentication tag
   ]
```

This object encodes to a total size of 24 bytes.

Figure 5 summarizes these results.

Scheme	seq+cid	TAG	Total Size	Overhead
COSE	11 B	8 B	44 bytes	25 bytes
mod-COSE	11 B	8 B	29 bytes	10 bytes
bound	11 B	8 B	24 bytes	5 bytes

Figure 5: Comparison of COSE, modified COSE and CBOR lower bound for AES-CCM.

B.4. Symmetric Encryption with Asymmetric Signature (SEAS)

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

The object in COSE encoding gives:

```
   [
     1,                                        # msg_type
     h'a16373657143112233',                    # protected:
                                                 {"seq": h'112233'}
     {},                                       # unprotected
     h'85024ba2010a6373657143112233a04081834
     0a201256363696448a1534e3c5fdc09bd40',     # payload:
                                            [2,
                                        h'a2010a6373657143112233',
                                        {}, h', [[h'',
                                        {1: -6,
                                             "cid": h'a1534e3c5fdc09bd'
                                             }, h'']]]
     [                                           # signatures
       [                                         # signature structure
         h'a201266363696448a1534e3c5fdc09bd', # protected:
                                                 {1: -7,
                                               "cid":h'a1534e3c5fdc09bd'}
         {},                                   # unprotected
         SIG                                   # 64-byte signature
       ]
     ]
   ]
```

The COSE object encodes to a total size of 134 bytes.

In the modified version of COSE defined in Appendix A.3, the equivalent COSE object would be:

```
   [
     6,                                        # msg_type
     h'a20b48a1534e3c5fdc09bd0c43112233',      # protected:
                                                 {11:h'a1534e3c5fdc09bd',
                                                  12:h'112233'}
     h'',                                      # ciphertext
     SIG                                       # 64-byte signature
   ]
```

This modified COSE object encodes to a total size of 86 bytes.

The low-bound CBOR encoding of this same object is encoded by:

```
   [
     h'112233',                                # seq
     h'a1534e3c5fdc09bd',                      # cid
     h'',                                      # ciphertext
     SIG                                       # 64-byte signature
   ]
```

This object encodes to a total size of 81 bytes.

Figure 6 summarizes these results.

Scheme	seq+cid	SIG	Total Size	Overhead
COSE	11 B	64 B	134 bytes	59 bytes
mod-COSE	11 B	64 B	86 bytes	11 bytes
bound	11 B	64 B	81 bytes	6 bytes

Figure 6: Comparison of nested AES-CCM within ECDSA (COSE) and combined AES-ECDSA (modified COSE and CBOR lower bound).

Appendix C. Object Security of Content (OSCON)

In this section we define how to only protect the payload/content of individual messages using the Secure Message format (Section 4) to comply with the requirements 1 and 2 in Section 2. This is referred to as Object Security of Content (OSCON).

Note that by only protecting the content of a message it may be verified by multiple recipients. For example, in the case of a proxy that supports caching, a recent response for a certain resource can be cached and used to serve multiple clients. Or, in a publish-subscribe setting, multiple subscribers can be served the same publication. The use of content protection also decouples the binding to the underlying transfer protocol, so the same protected content object can be freely move between CoAP, HTTP, BlueTooth or whatever application layer protocol.

The use of OSCON is signaled with the Content-Format/Media Type set to application/oscon (Section 10). Since the actual format of the content which is protected is lost, that information needs to be added to the message header or known to the recipient.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the SM format as described in this section. A CoAP client MAY request a response in the OSCON format by setting the option Accept to application/oscon.

In case of cipher suite for integrity protection only, the Authenticated Data SHALL be the concatenation of the SM Header and the CoAP Payload. If case of cipher suite for both encryption and integrity protection, then the AAD SHALL be the SM Header and the

Plaintext SHALL be the CoAP Payload. By default, cipher suites for encryption and integrity protection SHALL be used.

The SM SHALL be protected (encrypted) and verified (decrypted) as described in Section 5.1.3 (Section 5.2.2.1), including replay protection as described in Section 7.1.

Whereas in OSCOAP, the Context Identifier of the SM Header (Section 4.1) typically identifies the sending party, with OSCON (Appendix C) the Context Identifier may well identify the sender and resource.

C.1. Security Considerations of OSCON

OSCON (Appendix C) only protects payload and only gives replay protection (not freshness of response), but allows additional use cases such as point to multi-point interactions including publish-subscribe, reverse proxies and proxy caching of responses. In case of symmetric keys the receiver does not get data origin authentication, which requires a digital signature using a private asymmetric key.

OSCON SHALL NOT be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected. The request for a response in OSCON using the CoAP option Accept set to "application/oscon" is not secured since OSCON does not integrity protect any options. Hence the exchange of OSCON request-response messages is vulnerable to a man-in-the-middle attack where response is exchanged for another response, but since there is replay protection only messages with higher sequence numbers will be accepted.

Blockwise transfers in CoAP as defined in [I-D.ietf-core-block] can be applied with OSCON, i.e. the entire payload is encapsulated in a Secure Message which is partitioned into blocks which are sent with unprotected CoAP. The receiver is able to verify the integrity of the payload but only after the last block containing the signature/MAC is received, and if the verification fails the entire message needs to be resent. However, if the verification succeeds, then the transmission in OSCON has less computational and packet overhead since only one signature/MAC was generated and sent. As CoAP blockwise transfer with OSCON is prone to Denial of Service attacks, it should only be used for exchanges where this threat can be mitigated, for example within a local area network where link-layer security is activated.

Appendix D. Examples

This section gives examples of how to use the Object-Security option and the message formats defined in this memo.

D.1. CoAP Message Protection

This section illustrates Object Security of CoAP (OSCOAP). The message exchange assumes there is a security context established between client and server. One key is used for each direction of the message transfer. The intermediate node detects that the CoAP message contains an OSCOAP object (Object-Security option is set) and thus forwards the message as it cannot serve a cached response.

D.1.1. Integrity Protection of CoAP Message Exchange

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Object-Security option. The example illustrates a client closing a lock (PUT 1) and getting a confirmation that the lock is closed. Code, Uri-Path and Payload of the request and Code of the response are integrity protected (and other message fields, see Section 6.1 and Section 6.2).

```
   Client  Proxy  Server
      |      |      |
      |      |      |
      |      |      |
      +----->|      |      Code: 0.03 (PUT)
      | PUT  |      |     Token: 0x8c
      |      |      |  Uri-Path: lock
      |      |      |  Object-Security:
      |      |      |   Payload: ["seq":"142",
      |      |      |            "cid":"a1534e3c5fdc09bd", 1, <Tag>]
      |      |      |
      |      +----->|      Code: 0.03 (PUT)
      |      | PUT  |    Token: 0x7b
      |      |      |  Uri-Path: lock
      |      |      |  Object-Security:
      |      |      |   Payload: ["seq":"142",
      |      |      |            "cid":"a1534e3c5fdc09bd", 1, <Tag>]
      |      |      |
      |      |<-----+      Code: 2.04 (Changed)
      |      | 2.04 |     Token: 0x7b
      |      |      |   Object-Security: ["seq":"a6",
      |      |      |            "cid":"5fdc09bda1534e3c", , <Tag>]
      |      |      |
      |<-----+      |      Code: 2.04 (Changed)
      | 2.04 |      |     Token: 0x8c
      |      |      |   Object-Security: ["seq":"a6",
      |      |      |            "cid":"5fdc09bda1534e3c", , <Tag>]
      |      |      |
```

Figure 7: CoAP PUT protected with OSCOAP

Since the request message (PUT) supports payload, the OSCOAP object is carried in the CoAP payload. Since the response message (Changed) does not supports payload the Object-Security option carries the OSCOAP object.

The Header contains Sequence Number ("seq":"a6") and Context Identifier ("cid":"5fdc09bda1534e3c"), the latter is an identifier indicating which security context was used to integrity protect the message, and may be used as an identifier for a secret key or a public key. (It may e.g. be the hash of a public key.)

The server and client can verify that the Sequence Number has not been received and used with this key before. With OSCOAP, the client additionally verifies the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 7).

This example deviates from encryption by default (see Section 8) just to illustrate the case of Integrity Protection only. If there is no compelling reason why the CoAP message should be in plaintext, then it MUST be encrypted.

D.1.2. Additional Encryption of CoAP Message

Here is an example of a GET request/response message exchange passing an intermediate node protected with the Enc option. The example illustrates a client requesting a blood sugar measurement resource (GET /glucose) and receiving the value 220 mg/dl. Uri-Path and Payload are encrypted and integrity protected. Code is integrity protected only (see Section 6.1 and Section 6.2).

```
   Client  Proxy  Server
      |       |       |
      |       |       |
      |       |       |
      +----->|       |      Code: 0.01 (GET)
      | GET   |       |     Token: 0x83
      |       |       |   Object-Security: ["seq":"15b7",
      |       |       |            "cid":"34e3c5fdca1509bd",
      |       |       |            {"glucose"}, <Tag>]
      |       |       |
      |       +----->|      Code: 0.01 (GET)
      |       | GET   |     Token: 0xbe
      |       |       |   Object-Security: ["seq":"15b7",
      |       |       |            "cid":"34e3c5fdca1509bd",
      |       |       |            {"glucose"}, <Tag>]
      |       |       |
      |       |       |
      |       |<-----+      Code: 2.05 (Content)
      |       | 2.05 |     Token: 0xbe
      |       |       |   Object-Security:
      |       |       |   Payload: ["seq":"32c9",
      |       |       |            "cid":"c09bda155fd34e3c",
      |       |       |            {220}, <Tag>]
      |       |       |
      |<-----+       |      Code: 2.05 (Content)
      | 2.05 |       |     Token: 0x83
      |       |       |   Object-Security:
      |       |       |   Payload: ["seq":"32c9",
      |       |       |            "cid":"c09bda155fd34e3c",
      |       |       |            {220}, <Tag>]
      |       |       |
```

Figure 8: CoAP GET protected with OSCOAP. The bracket { ... } indicates encrypted data.

Since the request message (GET) does not support payload, the OSCOAP object is carried in the Object-Security option. Since the response message (Content) supports payload, the Object-Security option is empty and the OSCOAP object is carried in the payload.

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

The server and client can verify that the Sequence Number has not been received and used with this key before, and the client additionally verifies the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 7).

D.2. Payload Protection

This section gives examples that illustrate Object Security of Content (OSCON), see Appendix C). The assumption here is that only the intended receiver(s) has the relevant security context related to the resource. In case of a closed group of recipients of the same object, e.g. in Information-Centric Networking or firmware update distribution, it may be necessary to support symmetric key encryption in combination with digital signature.

D.2.1. Proxy Caching

This example outlines how a proxy forwarding request and response of one client can cache a response whose payload is a OSCON object, and serve this response to another client request, such that both clients can verify integrity and non-replay.

```
   Client1 Proxy  Server

      |      |      |
      |      |      |
      +----->|      |     Code: 0.01 (GET)
      | GET  |      |    Token: 0x83
      |      |      | Proxy-Uri: example.com/temp
      |      |      |
      |      |      |
      |      +----->|     Code: 0.01 (GET)
      |      | GET  |    Token: 0xbe
      |      |      |  Uri-Host: example.com
      |      |      |  Uri-Path: temp
      |      |      |
      |      |      |
      |      |<-----+     Code: 2.05 (Content)
      |      | 2.05 |    Token: 0xbe
      |      |      |   Payload: ["seq":"15b7",
      |      |      |            "cid":"c09bda155fd34e3c",
      |      |      |            "471 F", <Tag>]
      |      |      |
      |<-----+      |     Code: 2.05 (Content)
      | 2.05 |      |    Token: 0x83
      |      |      |   Payload: ["seq":"15b7",
      |      |      |            "cid":"c09bda155fd34e3c",
             |      |            "471 F", <Tag>]
   Client2   |      |
             |      |
      |      |      |
      |      |      |
      +----->|      |     Code: 0.01 (GET)
      | GET  |      |    Token: 0xa1
      |      |      | Proxy-Uri: example.com/temp
      |      |      |
      |<-----+      |     Code: 2.05 (Content)
      | 2.05 |      |    Token: 0xa1
      |      |      |   Payload: ["seq":"15b7",
      |      |      |            "cid":"c09bda155fd34e3c",
      |      |      |            "471 F", <Tag>]
```

Figure 9: Proxy caching protected with Object Security of Content (OSCON)

D.2.2. Publish-Subscribe

This example outlines a publish-subscribe setting where the payload is encrypted, integrity and replay protected end-to-end between Publisher and Subscriber. The example applies for example to closed

user groups of a single data source and illustrates a subscription registration and a later publication of birch pollen count of 300 per cubic meters. The PubSub Broker can define the Observe count arbitrarily (as could any intermediary node, even in OSCOAP), but cannot manipulate the Sequence Number without being possible to detect.

```
   Sub-    PubSub- Pub-
   scriber Broker  lisher

      |      |      |
      +----->|      |      Code: 0.01 (GET)
      | GET  |      |     Token: 0x72
      |      |      |  Uri-Path: ps
      |      |      |  Uri-Path: birch-pollen
      |      |      |   Observe: 0 (register)
      |      |      |
      |      |      |
      |<-----+      |      Code: 2.05 (Content)
      | 2.05 |      |     Token: 0x72
      |      |      |   Observe: 1
      |      |      |   Payload: ["seq":"15b7",
      |      |      |            "cid":"c09bda155fd34e3c",
      |      |      |            {"270"}, <Tag>]
      |      |      |
      |      |      |
      |      |      |
      |      |<-----+      Code: 0.03 (PUT)
      |      | PUT  |     Token: 0x1f
      |      |      |  Uri-Path: ps
      |      |      |  Uri-Path: birch-pollen
      |      |      |   Payload: ["seq":"15b8",
      |      |      |            "cid":"c09bda155fd34e3c",
      |      |      |            {"300"}, <Tag>]
      |      |      |
      |      +----->|      Code: 2.04 (Changed)
      |      | 2.04 |     Token: 0x1f
      |      |      |
      |      |      |
      |<-----+      |      Code: 2.05 (Content)
      | 2.05 |      |     Token: 0x72
      |      |      |   Observe: 2
      |      |      |   Payload: ["seq":"15b8",
      |      |      |            "cid":"c09bda155fd34e3c",
      |      |      |            {"300"}, <Tag>]
```

Figure 10: Publish-subscribe protected with OSCON. The bracket { ... } indicates encrypted data.

This example deviates from encryption by default (see Section 8) just to illustrate Integrity Protection only in the case of OSCON. If there is no compelling reason why the payload should be in plaintext, then encryption MUST be used.

D.2.3. Transporting Authorization Information

This example outlines the transportation of authorization information from a node producing (Authorization Server, AS) to a node consuming (Resource Server, RS) such information. Authorization information may for example be an authorization decision with respect to a Client (C) accessing a Resource to be enforced by RS, see e.g. [I-D.ietf-ace-actors] or [I-D.seitz-ace-core-authz]. Here, C is clearly not trusted with modifying the information, but may need to be involved in mediating the authorization information to the RS, for example, because AS and RS does not have direct connectivity. So end-to-end security is required and object security ("access tokens") is the natural candidate.

This example considers the authorization information to be encapsulated in a OSCON object, generated by AS. How C accesses the OSCON object is out of scope for this example, it may e.g. be using CoAP. C then requests RS to configure the authorization information in the OSCON object by doing POST to /authz-info. This particular resource has a default access policy that only new messages signed by AS are authorized. RS thus verifies the integrity and sequence number by using the existing security context for the AS, and responds accordingly, a) or b), see Figure 11.

```
   Authz           Resource
   Server  Client  Server
      |      |      |
      |      |      |     Client accesses Access Token:
      +- - ->|      |     ["seq":"142",
      |      |      |     "cid":"c09bda1534e3c5fdc09bd",
      |      |      |            <AuthzInfo>, <Tag>]
      |      |      |
      |      +----->|      Code: 0.02 (POST)
      |      | POST |     Token: 0xac
      |      |      |  Uri-Path: authz-info
      |      |      |   Payload: ["seq":"142",
      |      |      |            "cid":"c09bda1534e3c5fdc09bd",
      |      |      |            <AuthzInfo>, <Tag>]

   a)
      |      |      |
      |      |<-----+      Code: 2.04 (Changed)
      |      | 2.04 |     Token: 0xac
      |      |      |

   b)
      |      |      |
      |      |<-----+      Code: 4.01 (Unauthorized)
      |      | 4.01 |     Token: 0xac
      |      |      |
```

Figure 11: Protected Transfer of Access Token using OSCON

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson
Farogatan 6
Kista 16480
Sweden

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig@sics.se