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

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

June 29, 2015

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

Abstract

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

Table of Contents

1. Introduction

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

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

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

1.1 Terminology

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

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

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

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

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

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

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

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

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

2. Background

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

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

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

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

2. Protection against message replay;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Secure Message

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

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

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

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

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

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

4.1 Secure Message format

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

4.1.1 Secure Message Header

The following parameters SHALL be included in the SM Header:

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

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

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

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

4.1.2 Secure Message Body

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

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

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

4.1.2.1 Secure Signed Message Body

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

4.1.2.2 Secure Encrypted Message Body

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

4.1.3 Secure Message Tag

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

5. Message Protection

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

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

5.1 CoAP Message Protection (Mode:COAP)

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

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

5.1.1 The Sig Option

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

5.1.1.1 Option Structure

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

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

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

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

Table 1: The Sig Option

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

5.1.1.2 Integrity Protection and Verification

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

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

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

5.1.1.3 SSM Body

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

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

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

o the CoAP Payload (if any).

5.1.2 The Enc Option

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

5.1.2.1 Option Structure

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

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

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

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

Table 2: The Enc Option

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

depends on actual message format selected and is TBD.

5.1.2.2 Encryption and Decryption

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

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

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

5.1.2.3 SEM Body

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

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

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

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

5.1.2.4 CoAP Message with Enc Option

An unprotected CoAP message is encrypted and integrity protected by

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

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

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

- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message, and the Enc option. The options shall be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).

- If the unprotected CoAP message has no Payload then the Enc option SHALL contain the SEM with Detached Content. If the unprotected CoAP message has Payload, then the SEM option SHALL be empty and the Payload of the CoAP message SHALL be the SEM with Detached Content. The Payload is prefixed by the one-byte Payload Marker (0xFF).

5.1.3 SM Tag

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

5.2 Payload Only Protection (Mode:PAYL)

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

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

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

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

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

5.3 Replay Protection and Freshness

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

5.3.1 Replay Protection

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

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

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

5.3.2 Freshness

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

6. Security Considerations

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

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

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

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

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

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

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

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

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

7. Privacy Considerations

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

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

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

8. IANA Considerations

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

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

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

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

Media Type: application/smpayl

Encoding: -

Id: 70

Reference: [this document]

9. Acknowledgements

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

10. References

10.1 Normative References

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

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

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

[RFC7258] Farrell, S., and H. Tschofenig, "Pervasive Monitoring Is an Attack", RFC 7258, May 2014.

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

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

10.2 Informative References

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

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

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

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

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

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

RFC 5652, September 2009.

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

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

Appendix A. Which CoAP Header Fields and Options to Protect

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

A.1 CoAP Header Fields

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

A.2 CoAP Options

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

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

A.2.1 Integrity Protection

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

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

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

intermediaries and hence SHALL be integrity protected.

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

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

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

A.2.1.1 Proxy-Scheme

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

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

A.2.1.2 Uri-*

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

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

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

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

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

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

A.2.2 Encryption

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

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

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

A.2.3 Summary

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

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

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

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

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

Table 3: Protected CoAP options in Mode=COAP.

Appendix B. JOSE Profile of SM

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

B.1 JWS as Secure Signed Message

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

A JWS is a Secure Message if the JOSE Header includes

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

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

B.2 JWE as Secure Encrypted Message

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

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

A JWE is a Secure Message if the JOSE Header includes

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

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

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

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

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

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

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

Appendix C. Compact Secure Message

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

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

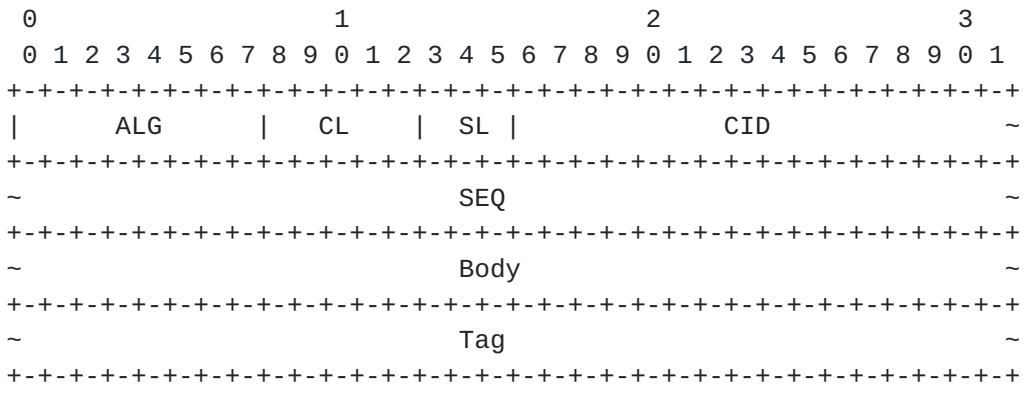


Figure 4: Compact Secure Message format

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

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

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

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

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

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

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

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

Appendix D. COSE Profile of SM

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

D.1 COSE_Sign or COSE_mac as Secure Signed Message

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

D.1.1 COSE_Sign as Secure Signed Message

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

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

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

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

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

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

D.1.2 COSE_mac as Secure Signed Message

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

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

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

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

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

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

The mandatory "tag" field contains the MAC value.

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

D.2 COSE_encrypt as Secure Encrypted Message

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

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

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

The "protected" field includes:

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

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

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

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

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

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

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

D.3 COSE optimizations

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

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

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

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

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

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

Appendix E. Comparison of message sizes

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

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

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

E.1 SSM: Message Authentication Code

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

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

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

The same object in COSE gives:

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

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

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

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

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

For CSM the same header is represented by 13 bytes.

Table 6 summarizes these results.

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

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

E.2 SSM: Digital Signature

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

For JWS the following header is used:

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

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

The same object in COSE gives:

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

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

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

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

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

For CSM the same header is represented by 13 bytes.

Table 7 summarizes these results.

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

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

E.3 SEM: Authenticated Encryption

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

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

The JWE header

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

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

The corresponding COSE object is:

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

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

Table 8 summarizes these results.

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

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

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

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

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

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

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

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

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

Table 9 summarizes these results.

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

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

E.4 SEM: Symmetric Encryption + Digital Signature

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

For JWS the following header is used:

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

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

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

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

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

The total size of the JWS object is 275 bytes.

The same object in COSE gives:

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

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

Table 10 summarizes these results.

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

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

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

The COSE object is represented as follows:

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

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

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

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

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

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

Table 11 summarizes these results.

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

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

Appendix F. Examples

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

F.1 CoAP Message Protection

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

F.1.1 Integrity Protection of CoAP Message Exchange

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

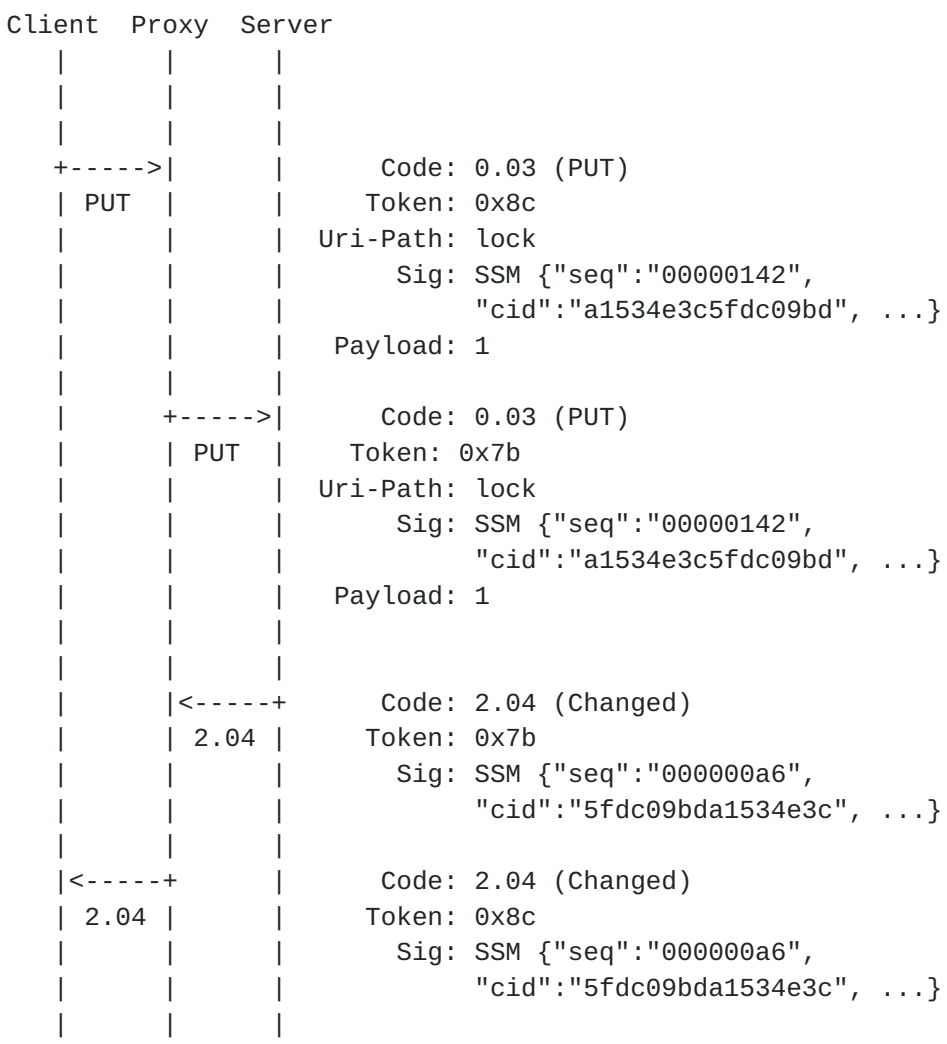


Figure 8: CoAP PUT protected with Sig/SSM (Mode:COAP)

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

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

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

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

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

F.1.2 Additional Encryption of CoAP Message

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

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

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

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

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

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

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

F.2 Payload Protection

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

F.2.1 Proxy Caching

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

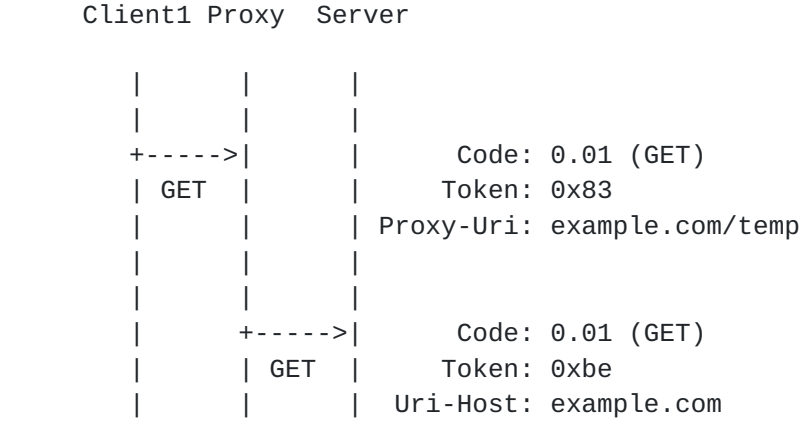
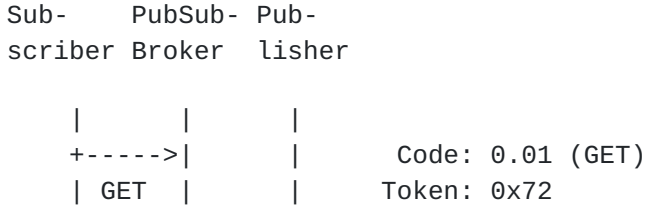


Figure 10: Proxy caching protected with SEM (Mode:PAYL)

F.2.2 Publish-Subscribe

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



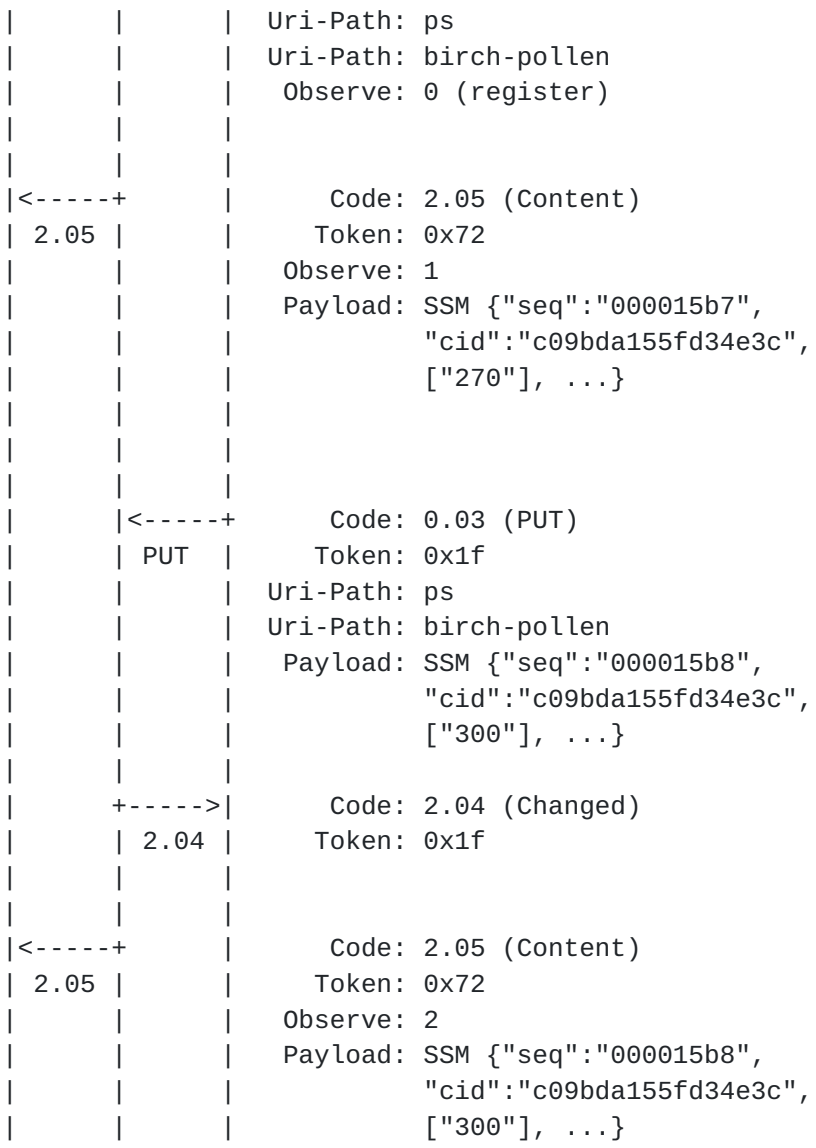


Figure 11: Publish-subscribe protected with SSM (Mode:PAYL)

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

F.2.3 Transporting Authorization Information

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

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

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

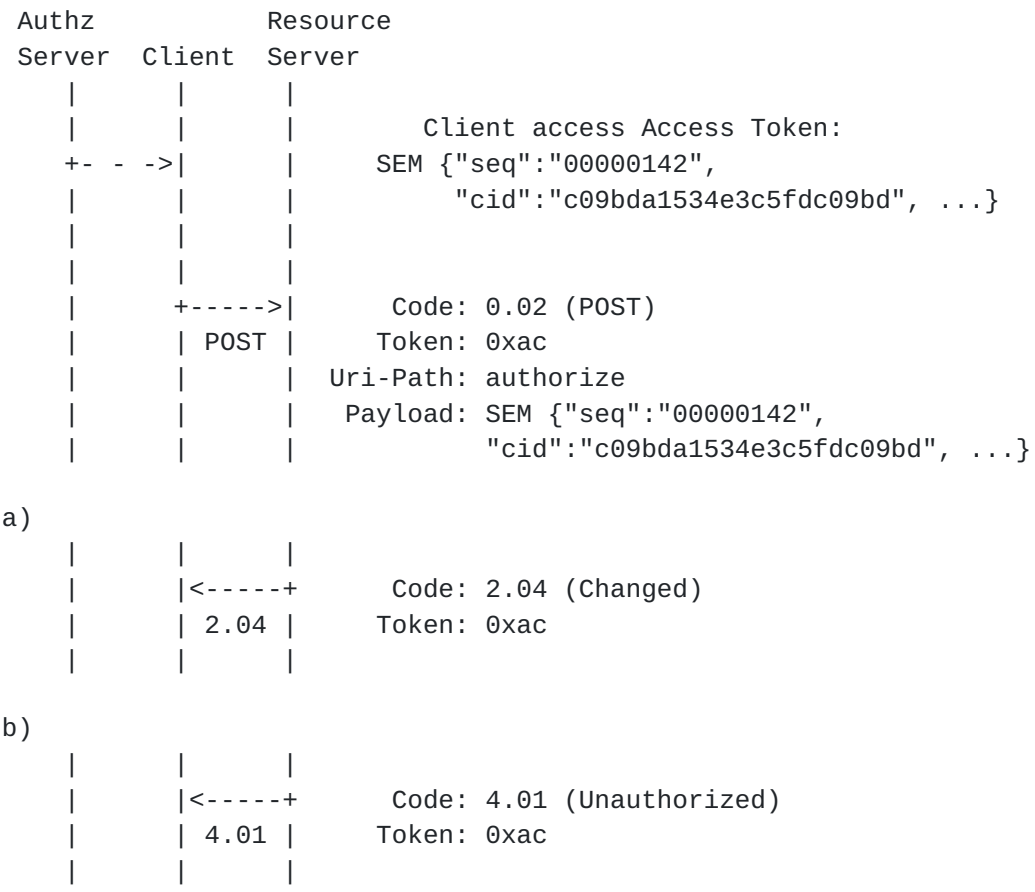


Figure 12: Protected Transfer of Access Token = SEM (Mode:PAYL)

Authors' Addresses

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

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

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

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