CoRE Working Group                                        G. Selander
Internet-Draft                                            J. Mattsson
Intended status: Standards Track                         F. Palombini
Expires: July 26, 2018                                   Ericsson AB
                                                            L. Seitz
                                                           RISE SICS
                                                    January 22, 2018

        **Object Security for Constrained RESTful Environments (OSCORE)**
                  **draft-ietf-core-object-security-08**

Abstract

   This document defines Object Security for Constrained RESTful
   Environments (OSCORE), a method for application-layer protection of
   the Constrained Application Protocol (CoAP), using CBOR Object
   Signing and Encryption (COSE).  OSCORE provides end-to-end protection
   between endpoints communicating using CoAP or CoAP-mappable HTTP.
   OSCORE is designed for constrained nodes and networks supporting a
   range of proxy operations, including translation between different
   transport protocols.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF).  Note that other groups may also distribute
   working documents as Internet-Drafts.  The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time.  It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on July 26, 2018.

publication of this document.  Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document.  Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

## 1.  Introduction

   The Constrained Application Protocol (CoAP) [RFC7252] is a web
   application protocol, designed for constrained nodes and networks
   [RFC7228], and may be mapped from HTTP [RFC8075].  CoAP specifies the
   use of proxies for scalability and efficiency and references DTLS
   ([RFC6347]) for security.  CoAP and HTTP proxies require (D)TLS to be
   terminated at the proxy.  The proxy therefore not only has access to
   the data required for performing the intended proxy functionality,
   but is also able to eavesdrop on, or manipulate any part of the
   message payload and metadata, in transit between the endpoints.  The
   proxy can also inject, delete, or reorder packets since they are no
   longer protected by (D)TLS.

   This document defines the Object Security for Constrained RESTful
   Environments (OSCORE) security protocol, protecting CoAP and CoAP-
   mappable HTTP requests and responses end-to-end across intermediary

nodes such as CoAP forward proxies and cross-protocol translators
including HTTP-to-CoAP proxies [RFC8075].  In addition to the core
CoAP features defined in [RFC7252], OSCORE supports Observe
[RFC7641], Blockwise [RFC7959], No-Response [RFC7967], and PATCH and
FETCH [RFC8132].  An analysis of end-to-end security for CoAP
messages through some types of intermediary nodes is performed in
[I-D.hartke-core-e2e-security-reqs].  OSCORE essentially protects the
RESTful interactions; the request method, the requested resource, the
message payload, etc. (see Section 4).  OSCORE does neither protect
the CoAP Messaging Layer nor the CoAP Token which may change between
the endpoints, and those are therefore processed as defined in
[RFC7252].  Additionally, since the message formats for CoAP over
unreliable transport [RFC7252] and for CoAP over reliable transport
[I-D.ietf-core-coap-tcp-tls] differ only in terms of CoAP Messaging
Layer, OSCORE can be applied to both unreliable and reliable
transports (see Figure 1).

```
            +-----------------------------------+
            |            Application             |
            +-----------------------------------+
            +-----------------------------------+  \
            |  Requests / Responses / Signaling |  |
            |-----------------------------------|  |
            |               OSCORE               |  | CoAP
            |-----------------------------------|  |
            | Messaging Layer / Message Framing |  |
            +-----------------------------------+  /
            +-----------------------------------+
            |           UDP / TCP / ...          |
            +-----------------------------------+
```
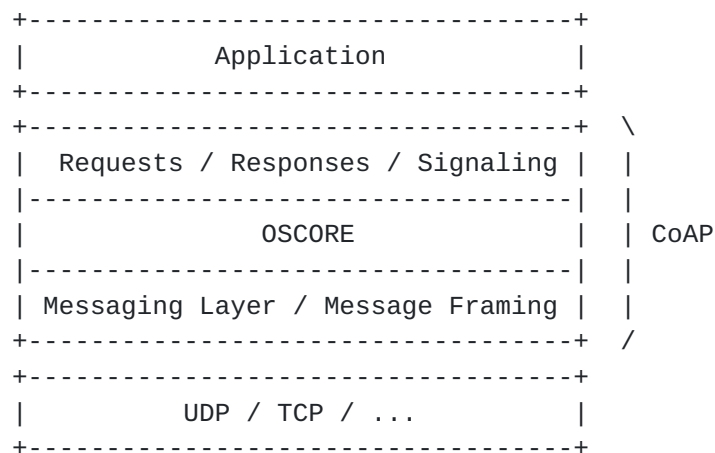
           Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE works in very constrained nodes and networks, thanks to its
small message size and the restricted code and memory requirements in
addition to what is required by CoAP.  Examples of the use of OSCORE
are given in Appendix A.  OSCORE does not depend on underlying
layers, and can be used anywhere where CoAP or HTTP can be used,
including non-IP transports (e.g., [I-D.bormann-6lo-coap-802-15-ie]).
OSCORE may be used together with (D)TLS over one or more hops in the
end-to-end path, e.g. with HTTPs in one hop and with plain CoAP in
another hop.

An extension of OSCORE may also be used to protect group
communication for CoAP [I-D.tiloca-core-multicast-oscoap].  The use
of OSCORE does not affect the URI scheme and OSCORE can therefore be
used with any URI scheme defined for CoAP or HTTP.  The application
decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-
band or with a key establishment protocol (see Section 3.2).  The
technical solution builds on CBOR Object Signing and Encryption
(COSE) [RFC8152], providing end-to-end encryption, integrity, replay
protection, and secure binding of response to request.  A compressed
version of COSE is used, as specified in Section 6.  The use of
OSCORE is signaled with the new Object-Security CoAP option or HTTP
header field, defined in Section 2 and Section 10.3.  The solution
transforms a CoAP/HTTP message into an "OSCORE message" before
sending, and vice versa after receiving.  The OSCORE message is a
CoAP/HTTP message related to the original message in the following
way: the original CoAP/HTTP message is translated to CoAP (if not
already in CoAP) and protected in a COSE object.  The encrypted
message fields of this COSE object are transported in the CoAP
payload/HTTP body of the OSCORE message, and the Object-Security
option/header field is included in the message.  A sketch of an
OSCORE message exchange in the case of the original message being
CoAP is provided in Figure 2).

```
      Client                                              Server
         |           OSCORE request - POST example.com:     |
         |             Header, Token,                        |
         |             Options: {Object-Security, ...},      |
         |             Payload: COSE ciphertext              |
         +------------------------------------------------->|
         |                                                   |
         |<-------------------------------------------------+
         |           OSCORE response - 2.04 (Changed):       |
         |             Header, Token,                         |
         |             Options: {Object-Security, ...},       |
         |             Payload: COSE ciphertext               |
         |                                                    |
```
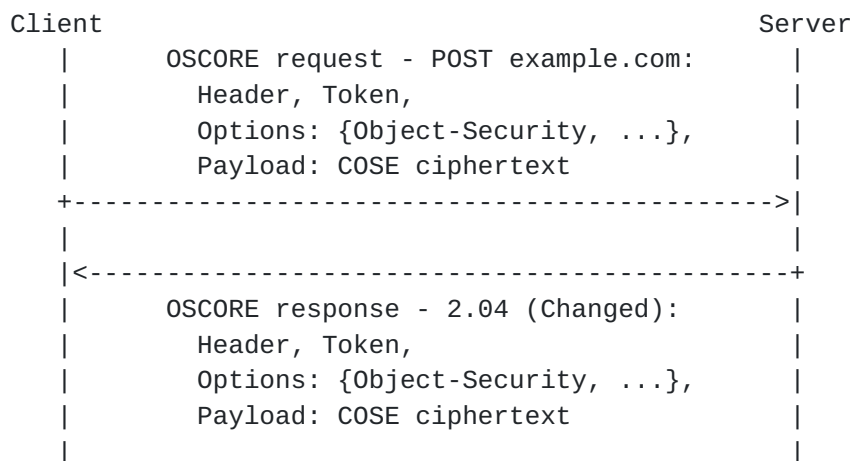
                 Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY only implement
the client part, MAY only implement the server part, or MAY only
implement one of the proxy parts.  OSCORE is designed to protect as
much information as possible while still allowing proxy operations
(Section 10).  It works with legacy CoAP-to-CoAP forward proxies
[RFC7252], but an OSCORE-aware proxy will be more efficient.  HTTP-
to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used
with OSCORE, as specified in Section 10.

## 1.1.  Terminology

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

Readers are expected to be familiar with the terms and concepts
described in CoAP [RFC7252], Observe [RFC7641], Blockwise [RFC7959],
COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl], and
constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end
path.  The concept "hop-by-hop" (as in "hop-by-hop encryption" or
"hop-by-hop fragmentation") opposed to "end-to-end", is used in this
document to indicate that the messages are processed accordingly in
the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote
that the message is not passed up to the CoAP Request/Response layer
(see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender
ID/Key, Recipient ID/Key, and Common IV are defined in Section 3.1.

## 2.  The CoAP Object-Security Option

The CoAP Object-Security option (see Figure 3) indicates that the
CoAP message is an OSCORE message and that it contains a compressed
COSE object (see Section 5 and Section 6).  The Object-Security
option is critical, safe to forward, part of the cache key, and not
repeatable.

```
+-----+---+---+---+---+-----------------+--------+--------+---------+
| No. | C | U | N | R | Name            | Format | Length | Default |
+-----+---+---+---+---+-----------------+--------+--------+---------+
| TBD | x |   |   |   | Object-Security |  (*)   | 0-255  | (none)  |
+-----+---+---+---+---+-----------------+--------+--------+---------+
   C = Critical,   U = Unsafe,   N = NoCacheKey,   R = Repeatable
   (*) See below.
```

Figure 3: The Object-Security Option

The Object-Security option includes the OSCORE flag bits (Section 6),
the Sender Sequence Number and the Sender ID when present
(Section 3).  The detailed format and length is specified in
Section 6.  If the OSCORE flag bits is all zero (0x00) the Option

value SHALL be empty (Option Length = 0).  An endpoint receiving a
CoAP message without payload, that also contains an Object-Security
option SHALL treat it as malformed and reject it.

A successful response to a request with the Object-Security option
SHALL contain the Object-Security option.  Whether error responses
contain the Object-Security option depends on the error type (see
Section 8).

A CoAP proxy SHOULD NOT cache a response to a request with an Object-
Security option, since the response is only applicable to the
original request (see Section 10.1).  As the compressed COSE Object
is included in the cache key, messages with the Object-Security
option will never generate cache hits.  For Max-Age processing (see
Section 4.2.3.1).

## 3.  The Security Context

OSCORE requires that client and server establish a shared security
context used to process the COSE objects.  OSCORE uses COSE with an
Authenticated Encryption with Additional Data (AEAD) algorithm for
protecting message data between a client and a server.  In this
section, we define the security context and how it is derived in
client and server based on a shared secret and a key derivation
function (KDF).

### 3.1.  Security Context Definition

The security context is the set of information elements necessary to
carry out the cryptographic operations in OSCORE.  For each endpoint,
the security context is composed of a "Common Context", a "Sender
Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and
verify messages received using the Recipient Context, both contexts
being derived from the Common Context and other data.  Clients and
servers need to be able to retrieve the correct security context to
use.

An endpoint uses its Sender ID (SID) to derive its Sender Context,
and the other endpoint uses the same ID, now called Recipient ID
(RID), to derive its Recipient Context.  In communication between two
endpoints, the Sender Context of one endpoint matches the Recipient
Context of the other endpoint, and vice versa.  Thus, the two
security contexts identified by the same IDs in the two endpoints are
not the same, but they are partly mirrored.  Retrieval and use of the
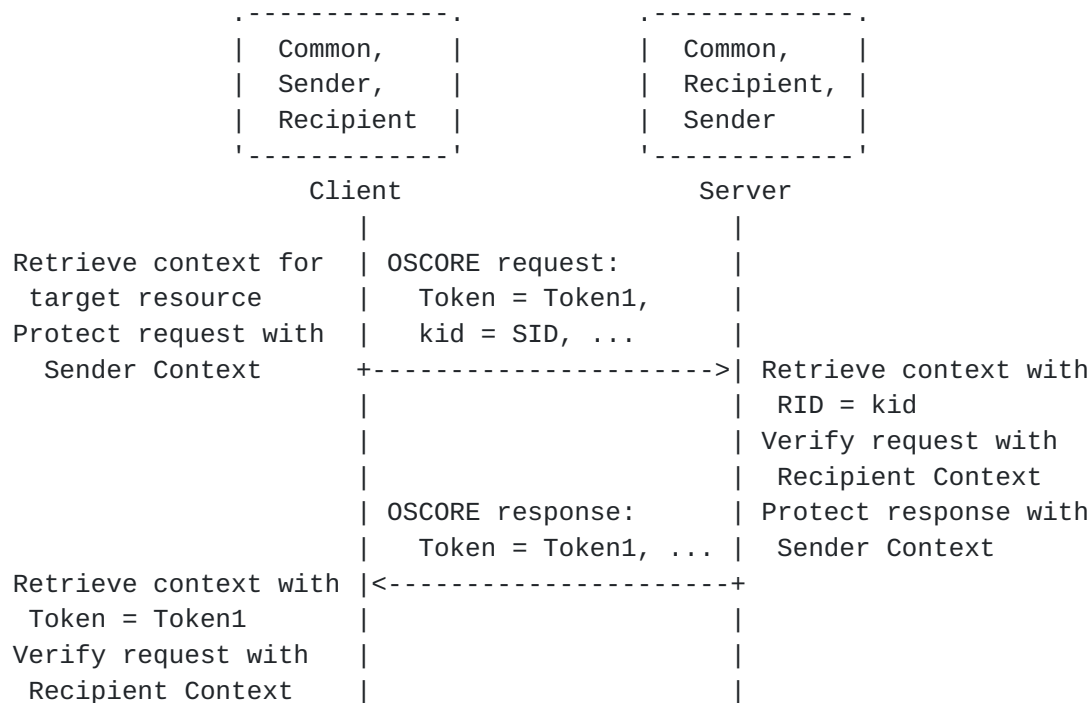security context are shown in Figure 4.

```
                .-------------.           .-------------.
                |  Common,    |           |  Common,    |
                |  Sender,    |           |  Recipient, |
                |  Recipient  |           |  Sender     |
                '-------------'           '-------------'
                    Client                   Server
                      |                        |
  Retrieve context for | OSCORE request:       |
   target resource     |   Token = Token1,     |
  Protect request with |   kid = SID, ...      |
    Sender Context      +--------------------->| Retrieve context with
                      |                        |  RID = kid
                      |                        | Verify request with
                      |                        |  Recipient Context
                      | OSCORE response:       | Protect response with
                      |   Token = Token1, ... | Sender Context
  Retrieve context with |<---------------------+
   Token = Token1     |                        |
  Verify request with  |                        |
   Recipient Context   |                        |
```

Figure 4: Retrieval and use of the Security Context

The Common Context contains the following parameters:

o  AEAD Algorithm.  The COSE AEAD algorithm to use for encryption.
   Its value is immutable once the security context is established.

o  Key Derivation Function.  The HMAC based HKDF [RFC5869] used to
   derive Sender Key, Recipient Key, and Common IV.

o  Master Secret.  Variable length, uniformly random byte string
   containing the key used to derive traffic keys and IVs.  Its value
   is immutable once the security context is established.

o  Master Salt.  Variable length byte string containing the salt used
   to derive traffic keys and IVs.  Its value is immutable once the
   security context is established.

o  Common IV.  Byte string derived from Master Secret and Master
   Salt.  Length is determined by the AEAD Algorithm.  Its value is
   immutable once the security context is established.

The Sender Context contains the following parameters:

o  Sender ID.  Byte string used to identify the Sender Context and to
   assure unique AEAD nonces.  Maximum length is determined by the

AEAD Algorithm.  Its value is immutable once the security context
is established.

o  Sender Key. Byte string containing the symmetric key to protect
   messages to send.  Derived from Common Context and Sender ID.
   Length is determined by the AEAD Algorithm.  Its value is
   immutable once the security context is established.

o  Sender Sequence Number.  Non-negative integer used by the sender
   to protect requests and Observe notifications.  Used as 'Partial
   IV' [RFC8152] to generate unique nonces for the AEAD.  Maximum
   value is determined by the AEAD Algorithm.

The Recipient Context contains the following parameters:

o  Recipient ID.  Byte string used to identify the Recipient Context
   and to assure unique AEAD nonces.  Maximum length is determined by
   the AEAD Algorithm.  Its value is immutable once the security
   context is established.

o  Recipient Key. Byte string containing the symmetric key to verify
   messages received.  Derived from Common Context and Recipient ID.
   Length is determined by the AEAD Algorithm.  Its value is
   immutable once the security context is established.

o  Replay Window (Server only).  The replay window to verify requests
   received.

An endpoint may free up memory by not storing the Common IV, Sender
Key, and Recipient Key, deriving them from the Master Key and Master
Salt when needed.  Alternatively, an endpoint may free up memory by
not storing the Master Secret and Master Salt after the other
parameters have been derived.

Endpoints MAY operate as both client and server and use the same
security context for those roles.  Independent of being client or
server, the endpoint protects messages to send using its Sender
Context, and verifies messages received using its Recipient Context.
The endpoints MUST NOT change the Sender/Recipient ID when changing
roles.  In other words, changing the roles does not change the set of
keys to be used.

## 3.2.  Establishment of Security Context Parameters

The parameters in the security context are derived from a small set
of input parameters.  The following input parameters SHALL be pre-
established:

o  Master Secret

o  Sender ID

o  Recipient ID

The following input parameters MAY be pre-established.  In case any
of these parameters is not pre-established, the default value
indicated below is used:

o  AEAD Algorithm

   *  Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)

o  Master Salt

   *  Default is the empty string

o  Key Derivation Function (KDF)

   *  Default is HKDF SHA-256

o  Replay Window Type and Size

   *  Default is DTLS-type replay protection with a window size of 32
      ([RFC6347])

All input parameters need to be known to and agreed on by both
endpoints, but the replay window may be different in the two
endpoints.  How the input parameters are pre-established, is
application specific.  The OSCORE profile of the ACE framework may be
used to establish the necessary input parameters
([I-D.ietf-ace-oscore-profile]), or a key exchange protocol such as
the TLS/DTLS handshake ([I-D.mattsson-ace-tls-oscore]) providing
forward secrecy.  Other examples of deploying OSCORE are given in
Appendix B.

## 3.2.1.  Derivation of Sender Key, Recipient Key, and Common IV

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

    output parameter = HKDF(salt, IKM, info, L)

where:

o  salt is the Master Salt as defined above

o  IKM is the Master Secret as defined above

o  info is a CBOR array consisting of:

```
info = [
    id : bstr,
    alg_aead : int / tstr,
    type : tstr,
    L : uint
]
```

where:

o  id is the Sender ID or Recipient ID when deriving keys and the
   empty string when deriving the Common IV.  The encoding is
   described in Section 5.

o  alg_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

o  type is "Key" or "IV".  The label is an ASCII string, and does not
   include a trailing NUL byte.

o  L is the size of the key/IV for the AEAD algorithm used, in bytes.

   For example, if the algorithm AES-CCM-16-64-128 (see Section 10.2 in
   [RFC8152]) is used, the integer value for alg_aead is 10, the value
   for L is 16 for keys and 13 for the Common IV.

## 3.2.2.  Initial Sequence Numbers and Replay Window

   The Sender Sequence Number is initialized to 0.  The supported types
   of replay protection and replay window length is application specific
   and depends on how OSCORE is transported, see Section 7.4.  The
   default is DTLS-type replay protection with a window size of 32
   initiated as described in Section 4.1.2.6 of [RFC6347].

## 3.3.  Requirements on the Security Context Parameters

   As collisions may lead to the loss of both confidentiality and
   integrity, Sender ID SHALL be unique in the set of all security
   contexts using the same Master Secret and Master Salt.  When a
   trusted third party assigns identifiers (e.g., using
   [I-D.ietf-ace-oauth-authz]) or by using a protocol that allows the
   parties to negotiate locally unique identifiers in each endpoint, the

Sender IDs can be very short.  The maximum length of Sender ID in
bytes equals the length of AEAD nonce minus 6.  For AES-CCM-16-64-128
the maximum length of Sender ID is 7 bytes.  Sender IDs MAY be
uniformly random distributed byte strings if the probability of
collisions is negligible.

If Sender ID uniqueness cannot be guaranteed by construction, Sender
IDs MUST be long uniformly random distributed byte strings such that
the probability of collisions is negligible.

To enable retrieval of the right Recipient Context, the Recipient ID
SHOULD be unique in the sets of all Recipient Contexts used by an
endpoint.  The Client MAY provide a 'kid context' parameter
(Section 5.1) to help the Server find the right context.

While the triple (Master Secret, Master Salt, Sender ID) MUST be
unique, the same Master Salt MAY be used with several Master Secrets
and the same Master Secret MAY be used with several Master Salts.

## 4.  Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from
an HTTP message) into an OSCORE message, and vice versa.  OSCORE
protects as much of the original message as possible while still
allowing certain proxy operations (see Section 10).  This section
defines how OSCORE protects the message fields and transfers them
end-to-end between client and server (in any direction).

The remainder of this section and later sections discuss the behavior
in terms of CoAP messages.  If HTTP is used for a particular hop in
the end-to-end path, then this section applies to the conceptual CoAP
message that is mappable to/from the original HTTP message as
discussed in Section 10.  That is, an HTTP message is conceptually
transformed to a CoAP message and then to an OSCORE message, and
similarly in the reverse direction.  An actual implementation might
translate directly from HTTP to OSCORE without the intervening CoAP
representation.

Protection of Signaling messages (Section 5 of
[I-D.ietf-core-coap-tcp-tls]) is specified in Section 4.4.  The other
parts of this section target Request/Response messages.

Message fields of the CoAP message may be protected end-to-end
between CoAP client and CoAP server in different ways:

o  Class E: encrypted and integrity protected,

o  Class I: integrity protected only, or

   o  Class U: unprotected.

   The sending endpoint SHALL transfer Class E message fields in the
   ciphertext of the COSE object in the OSCORE message.  The sending
   endpoint SHALL include Class I message fields in the Additional
   Authenticated Data (AAD) of the AEAD algorithm, allowing the
   receiving endpoint to detect if the value has changed in transfer.
   Class U message fields SHALL NOT be protected in transfer.  Class I
   and Class U message field values are transferred in the header or
   options part of the OSCORE message, which is visible to proxies.

   Message fields not visible to proxies, i.e., transported in the
   ciphertext of the COSE object, are called "Inner" (Class E).  Message
   fields transferred in the header or options part of the OSCORE
   message, which is visible to proxies, are called "Outer" (Class I or
   U).  There are currently no Class I options defined.

   An OSCORE message may contain both an Inner and an Outer instance of
   a certain CoAP message field.  Inner message fields are intended for
   the receiving endpoint, whereas Outer message fields are used to
   support proxy operations.  Inner and Outer message fields are
   processed independently.

## 4.1.  CoAP Payload

   The CoAP Payload, if present in the original CoAP message, SHALL be
   encrypted and integrity protected and is thus an Inner message field.
   See Figure 5.

```
                  +------------------+---+---+
                  | Field            | E | U |
                  +------------------+---+---+
                  | Payload          | x |   |
                  +------------------+---+---+

            E = Encrypt and Integrity Protect (Inner)
            U = Unprotected (Outer)

               Figure 5: Protection of CoAP Payload
```

   The sending endpoint writes the payload of the original CoAP message
   into the plaintext (Section 5.3) input to the COSE object.  The
   receiving endpoint verifies and decrypts the COSE object, and
   recreates the payload of the original CoAP message.

**4.2**.  **CoAP Options**

   A summary of how options are protected is shown in Figure 6.  Note
   that some options may have both Inner and Outer message fields which
   are protected accordingly.  The options which require special
   processing are labelled with asterisks.

```
              +-----+-----------------+---+---+
              | No. | Name            | E | U |
              +-----+-----------------+---+---+
              |   1 | If-Match        | x |   |
              |   3 | Uri-Host        |   | x |
              |   4 | ETag            | x |   |
              |   5 | If-None-Match   | x |   |
              |   6 | Observe         |   | * |
              |   7 | Uri-Port        |   | x |
              |   8 | Location-Path   | x |   |
              | TBD | Object-Security |   | * |
              |  11 | Uri-Path        | x |   |
              |  12 | Content-Format  | x |   |
              |  14 | Max-Age         | * | * |
              |  15 | Uri-Query       | x |   |
              |  17 | Accept          | x |   |
              |  20 | Location-Query  | x |   |
              |  23 | Block2          | * | * |
              |  27 | Block1          | * | * |
              |  28 | Size2           | * | * |
              |  35 | Proxy-Uri       |   | * |
              |  39 | Proxy-Scheme    |   | x |
              |  60 | Size1           | * | * |
              | 258 | No-Response     | * | * |
              +-----+-----------------+---+---+

          E = Encrypt and Integrity Protect (Inner)
          U = Unprotected (Outer)
          * = Special
```

                 Figure 6: Protection of CoAP Options

   Options that are unknown or for which OSCORE processing is not
   defined SHALL be processed as class E (and no special processing).
   Specifications of new CoAP options SHOULD define how they are
   processed with OSCORE.  A new COAP option SHOULD be of class E unless
   it requires proxy processing.

### 4.2.1.  Inner Options

Inner option message fields (class E) are used to communicate
directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields
present in the original CoAP message into the plaintext of the COSE
object (Section 5.3), and then remove the Inner option message fields
from the OSCORE message.

The processing of Inner option message fields by the receiving
endpoint is specified in Section 8.2 and Section 8.4.

### 4.2.2.  Outer Options

Outer option message fields (Class U or I) are used to support proxy
operations.

The sending endpoint SHALL include the Outer option message field
present in the original message in the options part of the OSCORE
message.  All Outer option message fields, including Object-Security,
SHALL be encoded as described in Section 3.1 of [RFC7252], where the
delta is the difference to the previously included Outer option
message field.

The processing of Outer options by the receiving endpoint is
specified in Section 8.2 and Section 8.4.

A procedure for integrity-protection-only of Class I option message
fields is specified in Section 5.4.  New CoAP options which are
repeatable and of class I MUST specify that proxies MUST NOT change
the order of the option's occurrences.

Note: There are currently no Class I option message fields defined.

### 4.2.3.  Special Options

Some options require special processing, marked with an asterisk '*'
in Figure 6; the processing is specified in this section.

### 4.2.3.1.  Max-Age

An Inner Max-Age message field is used to indicate the maximum time a
response may be cached by the client (as defined in [RFC7252]), end-
to-end from the server to the client, taking into account that the
option is not accessible to proxies.  The Inner Max-Age SHALL be
processed by OSCORE as specified in Section 4.2.1.

An Outer Max-Age message field is used to avoid unnecessary caching of OSCORE error responses at OSCORE unaware intermediary nodes.  A server MAY set a Class U Max-Age message field with value zero to OSCORE error responses described in Section 7.4, Section 8.2 and Section 8.4, which is then processed according to Section 4.2.2.

Successful OSCORE responses do not need to include an Outer Max-Age option since the responses are non-cacheable by construction (see Section 4.3).

### 4.2.3.2.  The Block Options

Blockwise [RFC7959] is an optional feature.  An implementation MAY support [RFC7252] and the Object-Security option without supporting Blockwise.  The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message fragmentation such that each fragment can be verified.  The Block options, when Outer message fields, enables hop-by-hop fragmentation of the OSCORE message.  Inner and Outer block processing may have different performance properties depending on the underlying transport.  The end-to-end integrity of the message can be verified both in case of Inner and Outer Blockwise provided all blocks are received.

### 4.2.3.2.1.  Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE.  In this case the Block options SHALL be processed by OSCORE as Inner options (Section 4.2.1).  The receiving CoAP endpoint SHALL process the OSCORE message according to Section 4.2.1 before processing Blockwise as defined in [RFC7959].

For concurrent Blockwise operations the sending endpoint MUST ensure that the receiving endpoint can distinguish between blocks from different operations.  One mechanism enabling this is specified in [I-D.ietf-core-echo-request-tag].

### 4.2.3.2.2.  Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.2.2) and not generated by the sending endpoint.  Note that the Outer Block options are neither encrypted nor integrity protected.  As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity.  Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a

maximum unfragmented message size (MAX_UNFRAGMENTED_SIZE) considering
the maximum size of message which can be handled by the endpoints.
Messages exceeding this size SHOULD be fragmented by the sending
endpoint using Inner Block options ([Section 4.2.3.2.1](#)).

An endpoint receiving an OSCORE message with an Outer Block option
SHALL first process this option according to [[RFC7959](#)], until all
blocks of the OSCORE message have been received, or the cumulated
message size of the blocks exceeds MAX_UNFRAGMENTED_SIZE.  In the
former case, the processing of the OSCORE message continues as
defined in this document.  In the latter case the message SHALL be
discarded.

Because of encryption of Uri-Path and Uri-Query, messages to the same
server may, from the point of view of a proxy, look like they also
target the same resource.  A proxy SHOULD mitigate a potential mix-up
of blocks from concurrent requests to the same server, for example
using the Request-Tag processing specified in Section 3.3.2 of
[[I-D.ietf-core-echo-request-tag](#)].

### [4.2.3.3](#).  **Proxy-Uri**

Proxy-Uri, when present, is split by OSCORE into class U options and
class E options, which are processed accordingly.  When Proxy-Uri is
used in the original CoAP message, Uri-* are not present [[RFC7252](#)].

The sending endpoint SHALL first decompose the Proxy-Uri value of the
original CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port, Uri-
Path, and Uri-Query options (if present) according to [Section 6.4 of
[RFC7252]](#).

Uri-Path and Uri-Query are class E options and SHALL be protected and
processed as Inner options ([Section 4.2.1](#)).

The Proxy-Uri option of the OSCORE message SHALL be set to the
composition of Proxy-Scheme, Uri-Host, and Uri-Port options (if
present) as specified in [Section 6.5 of [RFC7252]](#), and processed as
an Outer option of Class U ([Section 4.2.2](#)).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and
Uri-* options works by design for all CoAP URIs (see [Section 6 of
[RFC7252]](#)).  OSCORE-aware HTTP servers should not use the userinfo
component of the HTTP URI (as defined in [Section 3.2.1 of [RFC3986]](#)),
so that this type of replacement is possible in the presence of CoAP-
to-HTTP proxies.  In future documents specifying cross-protocol
proxying behavior using different URI structures, it is expected that
the authors will create Uri-* options that allow decomposing the
Proxy-Uri, and specify in which OSCORE class they belong.

An example of how Proxy-Uri is processed is given here.  Assume that
the original CoAP message contains:

o  Proxy-Uri = "coap://example.com/resource?q=1"

During OSCORE processing, Proxy-Uri is split into:

o  Proxy-Scheme = "coap"

o  Uri-Host = "example.com"

o  Uri-Port = "5683"

o  Uri-Path = "resource"

o  Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in
Section 4.2.1, and are thus encrypted and transported in the COSE
object.  The remaining options are composed into the Proxy-Uri
included in the options part of the OSCORE message, which has value:

o  Proxy-Uri = "coap://example.com"

See Sections 6.1 and 12.6 of [RFC7252] for more information.

### 4.2.3.4.  Observe

Observe [RFC7641] is an optional feature.  An implementation MAY
support [RFC7252] and the Object-Security option without supporting
[RFC7641].  The Observe option as used here targets the requirements
on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1).

In order for an OSCORE-unaware proxy to support forwarding of Observe
messages ([RFC7641]), there SHALL be an Outer Observe option, i.e.,
present in the options part of the OSCORE message.  The processing of
the CoAP Code for Observe messages is described in Section 4.3.

To secure the order of notifications, the client SHALL maintain a
Notification Number for each Observation it registers.  The
Notification Number is a non-negative integer containing the largest
Partial IV of the successfully received notifications for the
associated Observe registration (see Section 7.4).  The Notification
Number is initialized to the Partial IV of the first successfully
received notification response to the registration request.  In
contrast to [RFC7641], the received Partial IV MUST always be
compared with the Notification Number, which thus MUST NOT be

forgotten after 128 seconds.  The client MAY ignore the Observe
option value.

If the verification fails, the client SHALL stop processing the
response.

The Observe option in the CoAP request may be legitimately removed by
a proxy.  If the Observe option is removed from a CoAP request by a
proxy, then the server can still verify the request (as a non-Observe
request), and produce a non-Observe response.  If the OSCORE client
receives a response to an Observe request without an Outer Observe
value, then it MUST verify the response as a non-Observe response.
If the OSCORE client receives a response to a non-Observe request
with an Outer Observe value, it stops processing the message, as
specified in Section 8.4.

Clients can re-register observations to ensure that the observation
is still active and establish freshness again ([RFC7641]
Section 3.3.1).  When an OSCORE observation is refreshed, not only
the ETags, but also the partial IV (and thus the payload and Object-
Security option) change.  The server uses the new request's Partial
IV as the 'request_piv' of new responses.

### 4.2.3.5.  No-Response

No-Response is defined in [RFC7967].  Clients using No-Response MUST
set both an Inner (Class E) and an Outer (Class U) No-Response
option, with same value.

The Inner No-Response option is used to communicate to the server the
client's disinterest in certain classes of responses to a particular
request.  The Inner No-Response SHALL be processed by OSCORE as
specified in Section 4.2.1.

The Outer No-Response option is used to support proxy functionality,
specifically to avoid error transmissions from proxies to clients,
and to avoid bandwidth reduction to servers by proxies applying
congestion control when not receiving responses.  The Outer No-
Response option is processed according to Section 4.2.2.

In particular, step 8 of Section 8.4 is applied to No-Response.

Applications should consider that a proxy may remove the Outer No-
Response option from the request.  Applications using No-Response can
specify policies to deal with cases where servers receive an Inner
No-Response option only, which may be the result of the request
having traversed a No-Response unaware proxy, and update the
processing in Section 8.4 accordingly.  This avoids unnecessary error

responses to clients and bandwidth reductions to servers, due to No-Response unaware proxies.

**4.2.3.6.  Object-Security**

The Object-Security option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed.  The content in the Object-Security option is neither encrypted nor integrity protected as a whole but some part of the content of this option is protected (see Section 5.4).  "OSCORE within OSCORE" is not supported: If OSCORE processing detects an Object-Security option in the original CoAP message, then processing SHALL be stopped.

**4.3.  CoAP Header**

A summary of how the CoAP Header fields are protected is shown in Figure 7, including fields specific to CoAP over UDP and CoAP over TCP (marked accordingly in the table).

```
            +------------------+---+---+
            | Field            | E | U |
            +------------------+---+---+
            | Version (UDP)    |   | x |
            | Type (UDP)       |   | x |
            | Length (TCP)     |   | x |
            | Token Length     |   | x |
            | Code             | x |   |
            | Message ID (UDP) |   | x |
            | Token            |   | x |
            +------------------+---+---+

        E = Encrypt and Integrity Protect (Inner)
        U = Unprotected (Outer)
```

Figure 7: Protection of CoAP Header Fields

Most CoAP Header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP Header field Code is protected by OSCORE.  Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP
message into the plaintext of the COSE object (see Section 5.3).
After that, the Outer Code of the OSCORE message SHALL be set to 0.02
(POST) for requests without Observe option, to 0.05 (FETCH) for
requests with Observe option, and to 2.04 (Changed) for responses.
Using FETCH with Observe allows OSCORE to be compliant with the
Observe processing in OSCORE-unaware proxies.  The choice of POST and
FETCH ([RFC8132]) allows all OSCORE messages to have payload.

The receiving endpoint SHALL discard the Code in the OSCORE message
and write the Code of the plaintext in the COSE object (Section 5.3)
into the decrypted CoAP message.

The other CoAP Header fields are Unprotected (Class U).  The sending
endpoint SHALL write all other header fields of the original message
into the header of the OSCORE message.  The receiving endpoint SHALL
write the header fields from the received OSCORE message into the
header of the decrypted CoAP message.

## 4.4.  Signaling Messages

Signaling messages (CoAP Code 7.00-7.31) were introduced to exchange
information related to an underlying transport connection in the
specific case of CoAP over reliable transports
([I-D.ietf-core-coap-tcp-tls]).  The use of OSCORE for protecting
Signaling is application dependent.

OSCORE MAY be used to protect Signaling if the endpoints for OSCORE
coincide with the endpoints for the connection.  If OSCORE is used to
protect Signaling then:

o  Signaling messages SHALL be protected as CoAP Request messages,
   except in the case the Signaling message is a response to a
   previous Signaling message, in which case it SHALL be protected as
   a CoAP Response message.  For example, 7.02 (Ping) is protected as
   a CoAP Request and 7.03 (Pong) as a CoAP response.

o  The Outer Code for Signaling messages SHALL be set to 0.02 (POST),
   unless it is a response to a previous Signaling message, in which
   case it SHALL be set to 2.04 (Changed).

o  All Signaling options, except the Object-Security option, SHALL be
   Inner (Class E).

NOTE: Option numbers for Signaling messages are specific to the CoAP
Code (see Section 5.2 of [I-D.ietf-core-coap-tcp-tls]).

If OSCORE is not used to protect Signaling, Signaling messages SHALL
be unaltered by OSCORE.

## 5.  The COSE Object

This section defines how to use COSE [RFC8152] to wrap and protect
data in the original message.  OSCORE uses the untagged COSE_Encrypt0
structure with an Authenticated Encryption with Additional Data
(AEAD) algorithm.  The key lengths, IV length, nonce length, and
maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of
[RFC8152] is mandatory to implement.  For AES-CCM-16-64-128 the
length of Sender Key and Recipient Key is 128 bits, the length of
nonce and Common IV is 13 bytes.  The maximum Sender Sequence Number
is specified in Section 11.

As specified in [RFC5116], plaintext denotes the data that is
encrypted and integrity protected, and Additional Authenticated Data
(AAD) denotes the data that is integrity protected only.

The COSE Object SHALL be a COSE_Encrypt0 object with fields defined
as follows

o  The 'protected' field is empty.

o  The 'unprotected' field includes:

   *  The 'Partial IV' parameter.  The value is set to the Sender
      Sequence Number.  All leading zeroes SHALL be removed when
      encoding the Partial IV.  The value 0 encodes to the byte
      string 0x00.  This parameter SHALL be present in requests.  In
      case of Observe (Section 4.2.3.4) the Partial IV SHALL be
      present in responses, and otherwise the Partial IV SHOULD NOT
      be present in responses.  (A non-Observe example where the
      Partial IV is included in a response is provided in
      Section 7.5.2.)

   *  The 'kid' parameter.  The value is set to the Sender ID.  This
      parameter SHALL be present in requests and SHOULD NOT be
      present in responses.  An example where the Sender ID is
      included in a response is the extension of OSCORE to group
      communication [I-D.tiloca-core-multicast-oscoap].

   *  Optionally, a 'kid context' parameter as defined in
      Section 5.1.  This parameter MAY be present in requests and
      SHALL NOT be present in responses.

o  The 'ciphertext' field is computed from the secret key (Sender Key
   or Recipient Key), AEAD nonce (see Section 5.2), plaintext (see
   Section 5.3), and the Additional Authenticated Data (AAD) (see
   Section 5.4) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

## 5.1.  Kid Context

For certain use cases, e.g. deployments where the same 'kid' is used
with multiple contexts, it is necessary or favorable for the sender
to provide an additional identifier of the security material to use,
in order for the receiver to retrieve or establish the correct key.
The 'kid context' parameter is used to provide such additional input.
The 'kid context' is implicitly integrity protected, as manipulation
that leads to the wrong key (or no key) being retrieved which results
in an error, as described in Section 8.2.

A summary of the COSE header parameter 'kid context' defined above
can be found in Figure 8.

Some examples of relevant uses of kid context are the following:

o  If the client has an identifier in some other namespace which can
   be used by the server to retrieve or establish the security
   context, then that identifier can be used as kid context.  The kid
   context may be used as Master Salt (Section 3.1) for additional
   entropy of the security contexts (see for example
   [I-D.ietf-6tisch-minimal-security]).

o  In case of a group communication scenario
   [I-D.tiloca-core-multicast-oscoap], if the server belongs to
   multiple groups, then a group identifier can be used as kid
   context to enable the server to find the right security context.

```
+----------+--------+------------+----------------+-----------------+
|   name   | label  | value type | value registry | description     |
+----------+--------+------------+----------------+-----------------+
|   kid    | kidctx | bstr       |                | Identifies the  |
| context  |        |            |                | kid context     |
+----------+--------+------------+----------------+-----------------+
```

   Figure 8: Additional common header parameter for the COSE object

## 5.2.  Nonce

The AEAD nonce is constructed in the following way (see Figure 9):

1.  left-padding the Partial IV (in network byte order) with zeroes
    to exactly 5 bytes,

2.  left-padding the (Sender) ID of the endpoint that generated the
    Partial IV (in network byte order) with zeroes to exactly nonce
    length - 6 bytes,

3.  concatenating the size of the ID (S) with the padded ID and the
    padded Partial IV,

4.  and then XORing with the Common IV.

Note that in this specification only algorithms that use nonces equal
or greater than 7 bytes are supported.  The nonce construction with
S, ID of PIV generator, and Partial IV together with endpoint unique
IDs and encryption keys make it easy to verify that the nonces used
with a specific key will be unique.

When Observe is not used, the request and the response may use the
same nonce.  In this way, the Partial IV does not have to be sent in
responses, which reduces the size.  For processing instructions (see
Section 8).

```
        +---+----------------------+--+--+--+--+--+
        | S | ID of PIV generator  | Partial IV  |----+
        +---+----------------------+--+--+--+--+--+    |
                                                       |
        +-----------------------------------------+    |
        |                Common IV                |->(XOR)
        +-----------------------------------------+    |
                                                       |
        +-----------------------------------------+    |
        |                 Nonce                   |<---+
        +-----------------------------------------+
```
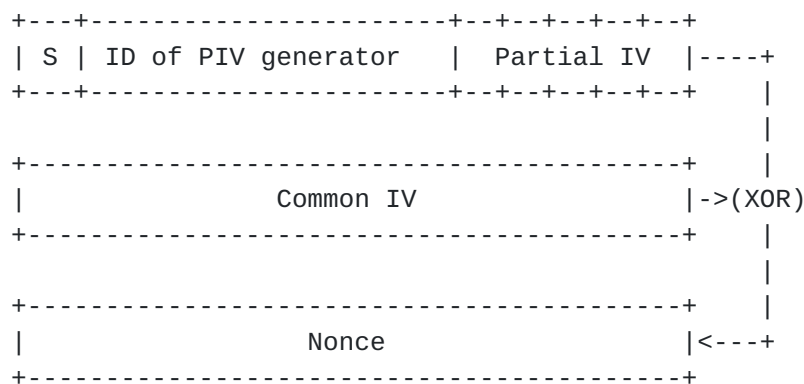
Figure 9: AEAD Nonce Formation

## 5.3.  Plaintext

The plaintext is formatted as a CoAP message without Header (see
Figure 10) consisting of:

o  the Code of the original CoAP message as defined in Section 3 of
   [RFC7252]; and

o  all Inner option message fields (see Section 4.2.1) present in the
   original CoAP message (see Section 4.2).  The options are encoded
   as described in Section 3.1 of [RFC7252], where the delta is the
   difference to the previously included Class E option; and

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |    Class E options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 (only if there
   is payload)
```

                        Figure 10: Plaintext

   NOTE: The plaintext contains all CoAP data that needs to be encrypted
   end-to-end between the endpoints.

## 5.4.  Additional Authenticated Data

   The external_aad SHALL be a CBOR array as defined below:

```
external_aad = [
   oscore_version : uint,
   [alg_aead : int / tstr],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr
]
```

   where:

o  oscore_version: contains the OSCORE version number.
   Implementations of this specification MUST set this field to 1.
   Other values are reserved for future versions.

o  alg_aead: contains the AEAD Algorithm from the security context
   used for the exchange (see Section 3.1).

o  request_kid: contains the value of the 'kid' in the COSE object of
   the request (see Section 5).

o  request_piv: contains the value of the 'Partial IV' in the COSE
   object of the request (see Section 5).

o  options: contains the Class I options (see Section 4.2.2) present
   in the original CoAP message encoded as described in Section 3.1
   of [RFC7252], where the delta is the difference to the previously
   included class I option.

NOTE: The format of the external_aad is for simplicity the same for
requests and responses, although some parameters, e.g. request_kid
need not be integrity protected in the requests.

## 6.  OSCORE Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines
very small message sizes with extensibility.  The CBOR Object Signing
and Encryption (COSE) [RFC8152] uses CBOR to create compact encoding
of signed and encrypted data.  COSE is however constructed to support
a large number of different stateless use cases, and is not fully
optimized for use as a stateful security protocol, leading to a
larger than necessary message expansion.  In this section, we define
a stateless compression mechanism, simply removing redundant
information from the COSE objects, which significantly reduces the
per-packet overhead.  The result of applying this mechanism to a COSE
object is called the "compressed COSE object".

### 6.1.  Encoding of the Object-Security Value

The value of the Object-Security option SHALL contain the OSCORE flag
bits, the Partial IV parameter, the kid context parameter (length and
value), and the kid parameter as follows:

```
     0 1 2 3 4 5 6 7 <--------- n bytes ------------->
    +-+-+-+-+-+-+-+-+--------------------------------
    |0 0 0|h|k|  n  |     Partial IV (if any) ...
    +-+-+-+-+-+-+-+-+--------------------------------

     <- 1 byte -> <------ s bytes ----->
    +------------+--------------------+-----------------+
    | s (if any) | kid context (if any) | kid (if any) ... |
    +------------+--------------------+-----------------+
```

                Figure 11: Object-Security Value

o  The first byte of flag bits encodes the following set of flags and
   the length of the Partial IV parameter:

* The three least significant bits encode the Partial IV length
  n.  If n = 0 then the Partial IV is not present in the
  compressed COSE object.  The values n = 6 and n = 7 are
  reserved.

* The fourth least significant bit is the kid flag, k: it is set
  to 1 if the kid is present in the compressed COSE object.

* The fifth least significant bit is the kid context flag, h: it
  is set to 1 if the compressed COSE object contains a kid
  context (see Section 5.1).

* The sixth to eighth least significant bits are reserved for
  future use.  These bits SHALL be set to zero when not in use.
  According to this specification, if any of these bits are set
  to 1 the message is considered to be malformed and
  decompression fails as specified in item 3 of Section 8.2.

o  The following n bytes encode the value of the Partial IV, if the
   Partial IV is present (n > 0).

o  The following 1 byte encode the length of the kid context
   (Section 5.1) s, if the kid context flag is set (h = 1).

o  The following s bytes encode the kid context, if the kid context
   flag is set (h = 1).

o  The remaining bytes encode the value of the kid, if the kid is
   present (k = 1).

Note that the kid MUST be the last field of the object-security
value, even in case reserved bits are used and additional fields are
added to it.

The length of the Object-Security option thus depends on the presence
and length of Partial IV, kid context, kid, as specified in this
section, and on the presence and length of the other parameters, as
defined in the separate documents.

## 6.2.  Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the
COSE object.

[6.3](#).  **Examples of Compressed COSE Objects**

[6.3.1](#).  **Examples: Requests**

   1.  Request with kid = 25 and Partial IV = 5

   Before compression (24 bytes):

   [
   h'',
   { 4:h'25', 6:h'05' },
   h'aea0155667924dff8a24e4cb35b9'
   ]

   After compression (17 bytes):

   Flag byte: 0b00001001 = 0x09

   Option Value: 09 05 25 (3 bytes)

   Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

   1.  Request with kid = empty string and Partial IV = 0

   After compression (16 bytes):

   Flag byte: 0b00001001 = 0x09

   Option Value: 09 00 (2 bytes)

   Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

   1.  Request with kid = empty string, Partial IV = 5, and kid context
       = 0x44616c656b

   After compression (22 bytes):

   Flag byte: 0b00011001 = 0x19

   Option Value: 19 05 05 44 61 6c 65 6b (8 bytes)

   Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

[6.3.2](#).  **Example: Response (without Observe)**

   Before compression (18 bytes):

```
[
h'',
{},
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00

Option Value: (0 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

### 6.3.3.  Example: Response (with Observe)

Before compression (21 bytes):

```
[
h'',
{ 6:h'07' },
h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01

Option Value: 01 07 (2 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 (14 bytes)

## 7.  Sequence Numbers, Replay, Message Binding, and Freshness

### 7.1.  Message Binding

In order to prevent response delay and mismatch attacks
[I-D.mattsson-core-coap-actuators] from on-path attackers and
compromised proxies, OSCORE binds responses to the requests by
including the kid and Partial IV of the request in the AAD of the
response.  The server therefore needs to store the kid and Partial IV
of the request until all responses have been sent.

### 7.2.  AEAD Nonce Uniqueness

An AEAD nonce MUST NOT be used more than once per AEAD key.  In order
to assure unique nonces, each Sender Context contains a Sender
Sequence Number used to protect requests, and - in case of Observe -

responses.  If messages are processed concurrently, the operation of
reading and increasing the Sender Sequence Number MUST be atomic.

The maximum Sender Sequence Number is algorithm dependent (see
Section 11), and no greater than 2^40 - 1.  If the Sender Sequence
Number exceeds the maximum, the endpoint MUST NOT process any more
messages with the given Sender Context.  The endpoint SHOULD acquire
a new security context (and consequently inform the other endpoint)
before this happens.  The latter is out of scope of this document.

## 7.3.  Freshness

For requests, OSCORE provides weak absolute freshness as the only
guarantee is that the request is not older than the security context.
For applications having stronger demands on request freshness (e.g.,
control of actuators), OSCORE needs to be augmented with mechanisms
providing freshness, for example as specified in
[I-D.ietf-core-echo-request-tag].

For responses, the message binding guarantees that a response is not
older than its request.  For responses without Observe, this gives
strong absolute freshness.  For responses with Observe, the absolute
freshness gets weaker with time, and it is RECOMMENDED that the
client regularly re-register the observation.

For requests, and responses with Observe, OSCORE also provides
relative freshness in the sense that the received Partial IV allows a
recipient to determine the relative order of responses.

## 7.4.  Replay Protection

In order to protect from replay of requests, the server's Recipient
Context includes a Replay Window.  A server SHALL verify that a
Partial IV received in the COSE object has not been received before.
If this verification fails the server SHALL stop processing the
message, and MAY optionally respond with a 4.01 Unauthorized error
message.  Also, the server MAY set an Outer Max-Age option with value
zero.  The diagnostic payload MAY contain the "Replay protection
failed" string.  The size and type of the Replay Window depends on
the use case and the protocol with which the OSCORE message is
transported.  In case of reliable and ordered transport from endpoint
to endpoint, e.g.  TCP, the server MAY just store the last received
Partial IV and require that newly received Partial IVs equals the
last received Partial IV + 1.  However, in case of mixed reliable and
unreliable transports and where messages may be lost, such a replay
mechanism may be too restrictive and the default replay window be
more suitable (see Section 3.2.2).

Responses to non-Observe requests are protected against replay as
they are cryptographically bound to the request.

In the case of Observe, a client receiving a notification SHALL
verify that the Partial IV of a received notification is greater than
the Notification Number bound to that Observe registration.  If the
verification fails, the client SHALL stop processing the response.
If the verification succeeds, the client SHALL overwrite the
corresponding Notification Number with the received Partial IV.

If messages are processed concurrently, the Partial IV needs to be
validated a second time after decryption and before updating the
replay protection data.  The operation of validating the Partial IV
and updating the replay protection data MUST be atomic.

## 7.5.  Losing Part of the Context State

To prevent reuse of the AEAD nonce with the same key, or from
accepting replayed messages, an endpoint needs to handle the
situation of losing rapidly changing parts of the context, such as
the request Token, Sender Sequence Number, Replay Window, and
Notification Numbers.  These are typically stored in RAM and
therefore lost in the case of an unplanned reboot.

After boot, an endpoint MAY reject to use existing security contexts
from before it booted and MAY establish a new security context with
each party it communicates.  However, establishing a fresh security
context may have a non-negligible cost in terms of, e.g., power
consumption.

After boot, an endpoint MAY use a partly persistently stored security
context, but then the endpoint MUST NOT reuse a previous Sender
Sequence Number and MUST NOT accept previously accepted messages.
Some ways to achieve this is described below:

### 7.5.1.  Sequence Number

To prevent reuse of Sender Sequence Numbers, an endpoint MAY perform
the following procedure during normal operations:

o  Each time the Sender Sequence Number is evenly divisible by K,
   where K is a positive integer, store the Sender Sequence Number in
   persistent memory.  After boot, the endpoint initiates the Sender
   Sequence Number to the value stored in persistent memory + K - 1.
   Storing to persistent memory can be costly.  The value K gives a
   trade-off between the number of storage operations and efficient
   use of Sender Sequence Numbers.

### 7.5.2.  Replay Window

To prevent accepting replay of previously received requests, the
server MAY perform the following procedure after boot:

o  For each stored security context, the first time after boot the
   server receives an OSCORE request, the server responds with the
   Echo option [I-D.ietf-core-echo-request-tag] to get a request with
   verifiable freshness.  The server MUST use its Partial IV when
   generating the AEAD nonce and MUST include the Partial IV in the
   response.

If the server using the Echo option can verify a second request as
fresh, then the Partial IV of the second request is set as the lower
limit of the replay window.

### 7.5.3.  Replay Protection of Observe Notifications

To prevent accepting replay of previously received notification
responses, the client MAY perform the following procedure after boot:

o  The client rejects notifications bound to the earlier
   registration, removes all Notification Numbers and re-registers
   using Observe.

## 8.  Processing

This section describes the OSCORE message processing.

### 8.1.  Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to
create an OSCORE request:

1.  Retrieve the Sender Context associated with the target resource.

2.  Compose the Additional Authenticated Data and the plaintext, as
    described in Section 5.4 and Section 5.3.

3.  Compute the AEAD nonce from the Sender ID, Common IV, and Partial
    IV (Sender Sequence Number in network byte order) as described in
    Section 5.2 and (in one atomic operation, see Section 7.2)
    increment the Sender Sequence Number by one.

4.  Encrypt the COSE object using the Sender Key. Compress the COSE
    Object as specified in Section 6.

5.  Format the OSCORE message according to Section 4.  The Object-
    Security option is added (see Section 4.2.2).

6.  Store the association Token - Security Context.  The client SHALL
    be able to find the Recipient Context from the Token in the
    response.

## 8.2.  Verifying the Request

A server receiving a request containing the Object-Security option
SHALL perform the following steps:

1.  Process Outer Block options according to [RFC7959], until all
    blocks of the request have been received (see Section 4.2.3.2).

2.  Discard the message Code and all non-special Inner option
    message fields (marked with 'x' in column E of Figure 6) present
    in the received message.  For example, an If-Match Outer option
    is discarded, but an Uri-Host Outer option is not discarded.

3.  Decompress the COSE Object (Section 6) and retrieve the
    Recipient Context associated with the Recipient ID in the 'kid'
    parameter.  If either the decompression or the COSE message
    fails to decode, or the server fails to retrieve a Recipient
    Context with Recipient ID corresponding to the 'kid' parameter
    received, then the server SHALL stop processing the request.
    If:

    *  either the decompression or the COSE message fails to decode,
       the server MAY respond with a 4.02 Bad Option error message.
       The server MAY set an Outer Max-Age option with value zero.
       The diagnostic payload SHOULD contain the string "Failed to
       decode COSE".

    *  the server fails to retrieve a Recipient Context with
       Recipient ID corresponding to the 'kid' parameter received,
       the server MAY respond with a 4.01 Unauthorized error
       message.  The server MAY set an Outer Max-Age option with
       value zero.  The diagnostic payload SHOULD contain the string
       "Security context not found".

4.  Verify the 'Partial IV' parameter using the Replay Window, as
    described in Section 7.4.

5.  Compose the Additional Authenticated Data, as described in
    Section 5.4.

6.   Compute the AEAD nonce from the Recipient ID, Common IV, and the
     'Partial IV' parameter, received in the COSE Object.

7.   Decrypt the COSE object using the Recipient Key.

     *  If decryption fails, the server MUST stop processing the
        request and MAY respond with a 4.00 Bad Request error
        message.  The server MAY set an Outer Max-Age option with
        value zero.  The diagnostic payload SHOULD contain the
        "Decryption failed" string.

     *  If decryption succeeds, update the Replay Window, as
        described in Section 7.

8.   For each decrypted option, check if the option is also present
     as an Outer option: if it is, discard the Outer.  For example:
     the message contains a Max-Age Inner and a Max-Age Outer option.
     The Outer Max-Age is discarded.

9.   Add decrypted code, options and payload to the decrypted
     request.  The Object-Security option is removed.

10.  The decrypted CoAP request is processed according to [RFC7252]

## 8.3.  Protecting the Response

If a CoAP response is generated in response to an OSCORE request, the
server SHALL perform the following steps to create an OSCORE
response.  Note that CoAP error responses derived from CoAP
processing (point 10. in Section 8.2) are protected, as well as
successful CoAP responses, while the OSCORE errors (point 3, 4, and 7
in Section 8.2) do not follow the processing below, but are sent as
simple CoAP responses, without OSCORE processing.

1.   Retrieve the Sender Context in the Security Context used to
     verify the request.

2.   Compose the Additional Authenticated Data and the plaintext, as
     described in Section 5.4 and Section 5.3.

3.   Compute the AEAD nonce

     *  If Observe is used, compute the nonce from the Sender ID,
        Common IV, and Partial IV (Sender Sequence Number in network
        byte order).  Then (in one atomic operation, see Section 7.2)
        increment the Sender Sequence Number by one.

* If Observe is not used, either the nonce from the request is used or a new Partial IV is used.

4.  Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in [Section 6](#).  If the AEAD nonce was constructed from a new Partial IV, this Partial IV MUST be included in the message.  If the AEAD nonce from the request was used, the Partial IV MUST NOT be included in the message.

5.  Format the OSCORE message according to [Section 4](#).  The Object-Security option is added (see [Section 4.2.2](#)).

## [8.4](#).  Verifying the Response

A client receiving a response containing the Object-Security option SHALL perform the following steps:

1.  Process Outer Block options according to [[RFC7959](#)], until all blocks of the OSCORE message have been received (see [Section 4.2.3.2](#)).

2.  Discard the message Code and all non-special Class E options from the message.  For example, ETag Outer option is discarded, Max-Age Outer option is not discarded.

3.  Retrieve the Recipient Context associated with the Token. Decompress the COSE Object ([Section 6](#)).  If either the decompression or the COSE message fails to decode, then go to 11.

4.  For Observe notifications, verify the received 'Partial IV' parameter against the corresponding Notification Number as described in [Section 7.4](#).  If the client receives a notification for which no Observe request was sent, then go to 11.

5.  Compose the Additional Authenticated Data, as described in [Section 5.4](#).

6.  Compute the AEAD nonce

    1.  If the Observe option and the Partial IV are not present in the response, the nonce from the request is used.

    2.  If the Observe option is present in the response, and the Partial IV is not present in the response, then go to 11.

3.  If the Partial IV is present in the response, compute the
    nonce from the Recipient ID, Common IV, and the 'Partial IV'
    parameter, received in the COSE Object.

7.  Decrypt the COSE object using the Recipient Key.

    *  If decryption fails, then go to 11.

    *  If decryption succeeds and Observe is used, update the
       corresponding Notification Number, as described in Section 7.

8.  For each decrypted option, check if the option is also present
    as an Outer option: if it is, discard the Outer.  For example:
    the message contains a Max-Age Inner and a Max-Age Outer option.
    The Outer Max-Age is discarded.

9.  Add decrypted code, options and payload to the decrypted
    request.  The Object-Security option is removed.

10. The decrypted CoAP response is processed according to [RFC7252]

11. (Optional) In case any of the previous erroneous conditions
    apply: the client SHALL stop processing the response.

An error condition occurring while processing a response in an
observation does not cancel the observation.  A client MUST NOT react
to failure in step 7 by re-registering the observation immediately.

## 9.  Web Linking

The use of OSCORE MAY be indicated by a target attribute "osc" in a
web link [RFC8288] to a resource.  This attribute is a hint
indicating that the destination of that link is to be accessed using
OSCORE.  Note that this is simply a hint, it does not include any
security context material or any other information required to run
OSCORE.

A value MUST NOT be given for the "osc" attribute; any present value
MUST be ignored by parsers.  The "osc" attribute MUST NOT appear more
than once in a given link-value; occurrences after the first MUST be
ignored by parsers.

## 10.  Proxy and HTTP Operations

RFC 7252 defines operations for a CoAP-to-CoAP proxy (see Section 5.7
of [RFC7252]) and for proxying between CoAP and HTTP (Section 10 of
[RFC7252]).  A more detailed description of the HTTP-to-CoAP mapping

is provided by [RFC8075].  This section describes the operations of
OSCORE-aware proxies.

## 10.1.  CoAP-to-CoAP Forwarding Proxy

OSCORE is designed to work with legacy CoAP-to-CoAP forward proxies
[RFC7252], but OSCORE-aware proxies MAY provide certain
simplifications as specified in this section.

Security requirements for forwarding are presented in Section 2.2.1
of [I-D.hartke-core-e2e-security-reqs].  OSCORE complies with the
extended security requirements also addressing Blockwise ([RFC7959])
and CoAP-mappable HTTP.  In particular caching is disabled since the
CoAP response is only applicable to the original CoAP request.  An
OSCORE-aware proxy SHALL NOT cache a response to a request with an
Object-Security option.  As a consequence, the search for cache hits
and CoAP freshness/Max-Age processing can be omitted.

Proxy processing of the (Outer) Proxy-Uri option is as defined in
[RFC7252].

Proxy processing of the (Outer) Block options is as defined in
[RFC7959] and [I-D.ietf-core-echo-request-tag].

Proxy processing of the (Outer) Observe option is as defined in
[RFC7641].  OSCORE-aware proxies MAY look at the Partial IV value
instead of the Outer Observe option.

## 10.2.  HTTP Processing

In order to use OSCORE with HTTP, an endpoint needs to be able to map
HTTP messages to CoAP messages (see [RFC8075]), and to apply OSCORE
to CoAP messages (as defined in this document).

A sending endpoint uses [RFC8075] to translate an HTTP message into a
CoAP message.  It then protects the message with OSCORE processing,
and add the Object-Security option (as defined in this document).
Then, the endpoint maps the resulting CoAP message to an HTTP message
that includes an HTTP header field named Object-Security, whose value
is:

o   "" (empty string) if the CoAP Object-Security option is empty, or

o   the value of the CoAP Object-Security option (Section 6.1) in
    base64url encoding (Section 5 of [RFC4648]) without padding (see
    [RFC7515] Appendix C for implementation notes for this encoding).

Note that the value of the HTTP body is the CoAP payload, i.e. the
OSCORE payload (Section 6.2).

The resulting message is an OSCORE message that uses HTTP.

A receiving endpoint uses [RFC8075] to translate an HTTP message into
a CoAP message, with the following addition.  The HTTP message
includes the Object-Security header field, which is mapped to the
CoAP Object-Security option in the following way.  The CoAP Object-
Security option value is:

o   empty if the value of the HTTP Object-Security header field is ""
    (empty string)

o   the value of the HTTP Object-Security header field decoded from
    base64url (Section 5 of [RFC4648]) without padding (see [RFC7515]
    Appendix C for implementation notes for this decoding).

Note that the value of the CoAP payload is the HTTP body, i.e. the
OSCORE payload (Section 6.2).

The resulting message is an OSCORE message that uses CoAP.

The endpoint can then verify the message according to the OSCORE
processing and get a verified CoAP message.  It can then translate
the verified CoAP message into a verified HTTP message.

## 10.3.  HTTP-to-CoAP Translation Proxy

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an
HTTP-to-CoAP proxy.  As requested in Section 1 of [RFC8075], this
section describes the HTTP mapping for the OSCORE protocol extension
of CoAP.

The presence of the Object-Security option, both in requests and
responses, is expressed in an HTTP header field named Object-Security
in the mapped request or response.  The value of the field is:

o   "" (empty string) if the CoAP Object-Security option is empty, or

o   the value of the CoAP Object-Security option (Section 6.1) in
    base64url encoding (Section 5 of [RFC4648]) without padding (see
    [RFC7515] Appendix C for implementation notes for this encoding).

The value of the body is the OSCORE payload (Section 6.2).

Example:

Mapping and notation here is based on "Simple Form" (Section 5.4.1.1
   of [RFC8075]).

 [HTTP request -- Before client object security processing]

   GET http://proxy.url/hc/?target_uri=coap://server.url/orders HTTP/1.1

   [HTTP request -- HTTP Client to Proxy]

     POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
     Object-Security: 09 25
     Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

   [CoAP request -- Proxy to CoAP Server]

     POST coap://server.url/
     Object-Security: 09 25
     Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

   [CoAP request -- After server object security processing]

     GET coap://server.url/orders

   [CoAP response -- Before server object security processing]

     2.05 Content
     Content-Format: 0
     Payload: Exterminate! Exterminate!

   [CoAP response -- CoAP Server to Proxy]

     2.04 Changed
     Object-Security: [empty]
     Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

   [HTTP response -- Proxy to HTTP Client]

     HTTP/1.1 200 OK
     Object-Security: "" (empty string)
     Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

   [HTTP response -- After client object security processing]

     HTTP/1.1 200 OK
     Content-Type: text/plain
     Body: Exterminate! Exterminate!

   Note that the HTTP Status Code 200 in the next-to-last message is the
   mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200
   in the last message is the mapping of the CoAP Code 2.05 (Content),
   which was encrypted within the compressed COSE object carried in the
   Body of the HTTP response.

10.4.  CoAP-to-HTTP Translation Proxy

   Section 10.1 of [RFC7252] describes the behavior of a CoAP-to-HTTP
   proxy.  RFC 8075 [RFC8075] does not cover this direction in any more
   detail and so an example instantiation of Section 10.1 of [RFC7252]
   is used below.

   Example:

   [CoAP request -- Before client object security processing]

     GET coap://proxy.url/
     Proxy-Uri=http://server.url/orders

   [CoAP request -- CoAP Client to Proxy]

     POST coap://proxy.url/
     Proxy-Uri=http://server.url/
     Object-Security: 09 25
     Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

   [HTTP request -- Proxy to HTTP Server]

     POST http://server.url/ HTTP/1.1
     Object-Security: 09 25
     Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

   [HTTP request -- After server object security processing]

     GET http://server.url/orders HTTP/1.1

   [HTTP response -- Before server object security processing]

     HTTP/1.1 200 OK
     Content-Type: text/plain
     Body: Exterminate! Exterminate!

   [HTTP response -- HTTP Server to Proxy]

     HTTP/1.1 200 OK
     Object-Security: "" (empty string)
     Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

```
[CoAP response - Proxy to CoAP Client]

  2.04 Changed
  Object-Security: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- After client object security processing]

  2.05 Content
  Content-Format: 0
  Payload: Exterminate! Exterminate!
```

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is
the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05
(Content) in the last message is the value that was encrypted within
the compressed COSE object carried in the Body of the HTTP response.

## 11.  Security Considerations

### 11.1.  End-to-end protection

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

(D)TLS protects hop-by-hop the entire message.  OSCORE protects end-
to-end all information that is not required for proxy operations (see
Section 4).  (D)TLS and OSCORE can be combined, thereby enabling end-
to-end security of the message payload, in combination with hop-by-
hop protection of the entire message, during transport between end-
point and intermediary node.  The CoAP messaging layer, including
header fields such as Type and Message ID, as well as CoAP message
fields Token and Token Length may be changed by a proxy and thus
cannot be protected end-to-end.  Error messages occurring during CoAP
processing are protected end-to-end.  Error messages occurring during
OSCORE processing are not always possible to protect, e.g. if the
receiving endpoint cannot locate the right security context.  It may
still be favorable to send an unprotected error message, e.g. to
prevent extensive retransmissions, so unprotected error messages are
allowed as specified.  Similar to error messages, signaling messages

are not always possible to protect as they may be intended for an
intermediary.  Hop-by-hop protection of signaling messages can be
achieved with (D)TLS.  Applications using unprotected error and
signaling messages need to consider the threat that these messages
may be spoofed.

## 11.2.  Security Context Establishment

The use of COSE to protect messages as specified in this document
requires an established security context.  The method to establish
the security context described in Section 3.2 is based on a common
shared secret material in client and server, which may be obtained,
e.g., by using the ACE framework [I-D.ietf-ace-oauth-authz].  An
OSCORE profile of ACE is described in [I-D.ietf-ace-oscore-profile].

## 11.3.  Replay Protection

Most AEAD algorithms require a unique nonce for each message, for
which the sender sequence numbers in the COSE message field 'Partial
IV' is used.  If the recipient accepts any sequence number larger
than the one previously received, then the problem of sequence number
synchronization is avoided.  With reliable transport, it may be
defined that only messages with sequence number which are equal to
previous sequence number + 1 are accepted.  The alternatives to
sequence numbers have their issues: very constrained devices may not
be able to support accurate time, or to generate and store large
numbers of random nonces.  The requirement to change key at counter
wrap is a complication, but it also forces the user of this
specification to think about implementing key renewal.

## 11.4.  Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD
algorithm.  The maximum sender sequence number SHALL be $2^{40} - 1$, or
any algorithm specific lower limit, after which a new security
context must be generated.  The mechanism to build the nonce
(Section 5.2) assumes that the nonce is at least 56 bit-long, and the
Partial IV is at most 40 bit-long.  The mandatory-to-implement AEAD
algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*.

The security level of a system with m Masters Keys of length k used
together with Master Salts with entropy n is $k + n - \log_2(m)$.
Similarly, the security level of a system with m AEAD keys of length
k used together with AEAD nonces of length n is $k + n - \log_2(m)$.
Security level here means that an attacker can recover one of the m
keys with complexity $2^{(k + n)} / m$.  Protection against such attacks
can be provided by increasing the size of the keys or the entropy of
the Master Salt.  The complexity of recovering a specific key is

still 2^k (assuming the Master Salt/AEAD nonce is public).  The
Master Secret, Sender Key, and Recipient Key MUST be secret, the rest
of the parameters MAY be public.  The Master Secret MUST be uniformly
random.

## 11.5.  Message Fragmentation

The Inner Block options enable the sender to split large messages
into OSCORE-protected blocks such that the receiving endpoint can
verify blocks before having received the complete message.  The Outer
Block options allow for arbitrary proxy fragmentation operations that
cannot be verified by the endpoints, but can by policy be restricted
in size since the Inner Block options allow for secure fragmentation
of very large messages.  A maximum message size (above which the
sending endpoint fragments the message and the receiving endpoint
discards the message, if complying to the policy) may be obtained as
part of normal resource discovery.

## 11.6.  Privacy Considerations

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

The unprotected options (Figure 6) may reveal privacy sensitive
information.  In particular Uri-Host SHOULD NOT contain privacy
sensitive information.  CoAP headers sent in plaintext allow, for
example, matching of CON and ACK (CoAP Message Identifier), matching
of request and responses (Token) and traffic analysis.

Unprotected error messages reveal information about the security
state in the communication between the endpoints.  Unprotected
signalling messages reveal information about the reliable transport
used on a leg of the path.  Using the mechanisms described in
Section 7.5 may reveal when a device goes through a reboot.  This can
be mitigated by the device storing the precise state of sender
sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the
message.  Applications may use a padding scheme to protect against
traffic analysis.  As an example, the strings "YES" and "NO" even if
encrypted can be distinguished from each other as there is no padding
supplied by the current set of encryption algorithms.  Some
information can be determined even from looking at boundary
conditions.  An example of this would be returning an integer between

0 and 100 where lengths of 1, 2 and 3 will provide information about
where in the range things are.  Three different methods to deal with
this are: 1) ensure that all messages are the same length.  For
example, using 0 and 1 instead of "yes" and "no". 2) Use a character
which is not part of the responses to pad to a fixed length.  For
example, pad with a space to three characters. 3) Use the PKCS #7
style padding scheme where m bytes are appended each having the value
of m.  For example, appending a 0 to "YES" and two 1's to "NO".  This
style of padding means that all values need to be padded.  Similar
arguments apply to other message fields such as resource names.

## 12.  IANA Considerations

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

Note to IANA: Please note all occurrences of "TBD" in this
specification should be assigned the same number.

### 12.1.  COSE Header Parameters Registry

The 'kid context' parameter is added to the "COSE Header Parameters
Registry":

o  Name: kid context

o  Label: kidctx

o  Value Type: bstr

o  Value Registry:

o  Description: kid context

o  Reference: Section 5.1 of this document

### 12.2.  CoAP Option Numbers Registry

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

```
            +--------+-----------------+------------------+
            | Number | Name            | Reference        |
            +--------+-----------------+------------------+
            |  TBD   | Object-Security | [[this document]] |
            +--------+-----------------+------------------+
```

## 12.3.  CoAP Signaling Option Numbers Registry

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

```
+------------+--------+--------------------+------------------+
| Applies to | Number | Name               | Reference        |
+------------+--------+--------------------+------------------+
| 7.xx       |  TBD   | Object-Security    | [[this document]] |
+------------+--------+--------------------+------------------+
```

## 12.4.  Header Field Registrations

The HTTP header field Object-Security is added to the Message Headers
registry:

```
+-------------------+----------+----------+------------------+
| Header Field Name | Protocol | Status   | Reference        |
+-------------------+----------+----------+------------------+
| Object-Security   | http     | standard | [[this document]] |
+-------------------+----------+----------+------------------+
```

## 13.  References

## 13.1.  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,
           <https://www.rfc-editor.org/info/rfc2119>.

[RFC4648]  Josefsson, S., "The Base16, Base32, and Base64 Data
           Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006,
           <https://www.rfc-editor.org/info/rfc4648>.

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

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

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

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

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

   [RFC8075]  Castellani, A., Loreto, S., Rahman, A., Fossati, T., and
              E. Dijk, "Guidelines for Mapping Implementations: HTTP to
              the Constrained Application Protocol (CoAP)", RFC 8075,
              DOI 10.17487/RFC8075, February 2017,
              <https://www.rfc-editor.org/info/rfc8075>.

   [RFC8132]  van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and
              FETCH Methods for the Constrained Application Protocol
              (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017,
              <https://www.rfc-editor.org/info/rfc8132>.

   [RFC8152]  Schaad, J., "CBOR Object Signing and Encryption (COSE)",
              RFC 8152, DOI 10.17487/RFC8152, July 2017,
              <https://www.rfc-editor.org/info/rfc8152>.

   [RFC8288]  Nottingham, M., "Web Linking", RFC 8288,
              DOI 10.17487/RFC8288, October 2017,
              <https://www.rfc-editor.org/info/rfc8288>.

## 13.2.  Informative References

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

   [I-D.hartke-core-e2e-security-reqs]
              Selander, G., Palombini, F., and K. Hartke, "Requirements
              for CoAP End-To-End Security", draft-hartke-core-e2e-
              security-reqs-03 (work in progress), July 2017.

   [I-D.ietf-6tisch-minimal-security]
              Vucinic, M., Simon, J., Pister, K., and M. Richardson,
              "Minimal Security Framework for 6TiSCH", draft-ietf-
              6tisch-minimal-security-04 (work in progress), October
              2017.

[I-D.ietf-ace-oauth-authz]
          Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
          H. Tschofenig, "Authentication and Authorization for
          Constrained Environments (ACE)", draft-ietf-ace-oauth-
          authz-09 (work in progress), November 2017.

[I-D.ietf-ace-oscore-profile]
          Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE
          profile of the Authentication and Authorization for
          Constrained Environments Framework", draft-ietf-ace-
          oscore-profile-00 (work in progress), December 2017.

[I-D.ietf-cbor-cddl]
          Birkholz, H., Vigano, C., and C. Bormann, "Concise data
          definition language (CDDL): a notational convention to
          express CBOR data structures", draft-ietf-cbor-cddl-00
          (work in progress), July 2017.

[I-D.ietf-core-coap-tcp-tls]
          Bormann, C., Lemay, S., Tschofenig, H., Hartke, K.,
          Silverajan, B., and B. Raymor, "CoAP (Constrained
          Application Protocol) over TCP, TLS, and WebSockets",
          draft-ietf-core-coap-tcp-tls-11 (work in progress),
          December 2017.

[I-D.ietf-core-echo-request-tag]
          Amsuess, C., Mattsson, J., and G. Selander, "Echo and
          Request-Tag", draft-ietf-core-echo-request-tag-00 (work in
          progress), October 2017.

[I-D.mattsson-ace-tls-oscore]
          Mattsson, J., "Using Transport Layer Security (TLS) to
          Secure OSCORE", draft-mattsson-ace-tls-oscore-00 (work in
          progress), October 2017.

[I-D.mattsson-core-coap-actuators]
          Mattsson, J., Fornehed, J., Selander, G., Palombini, F.,
          and C. Amsuess, "Controlling Actuators with CoAP", draft-
          mattsson-core-coap-actuators-03 (work in progress),
          October 2017.

[I-D.tiloca-core-multicast-oscoap]
          Tiloca, M., Selander, G., Palombini, F., and J. Park,
          "Secure group communication for CoAP", draft-tiloca-core-
          multicast-oscoap-04 (work in progress), October 2017.

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

   [RFC5116]  McGrew, D., "An Interface and Algorithms for Authenticated
              Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008,
              <https://www.rfc-editor.org/info/rfc5116>.

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

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

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

   [RFC7967]  Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T.
              Bose, "Constrained Application Protocol (CoAP) Option for
              No Server Response", RFC 7967, DOI 10.17487/RFC7967,
              August 2016, <https://www.rfc-editor.org/info/rfc7967>.

## Appendix A.  Scenario examples

   This section gives examples of OSCORE, targeting scenarios in
   Section 2.2.1.1 of [I-D.hartke-core-e2e-security-reqs].  The message
   exchanges are made, based on the assumption that there is a security
   context established between client and server.  For simplicity, these
   examples only indicate the content of the messages without going into
   detail of the (compressed) COSE message format.

## A.1.  Secure Access to Sensor

   This example illustrates a client requesting the alarm status from a
   server.

```
     Client  Proxy  Server
       |      |       |
      +------>|       |              Code: 0.02 (POST)
      | POST  |       |             Token: 0x8c
      |       |       | Object-Security: [kid:5f,Partial IV:42]
      |       |       |         Payload: {Code:0.01,
      |       |       |                   Uri-Path:"alarm_status"}
      |       |       |
      |      +------>|              Code: 0.02 (POST)
      |      | POST  |             Token: 0x7b
      |      |       | Object-Security: [kid:5f,Partial IV:42]
      |      |       |         Payload: {Code:0.01,
      |      |       |                   Uri-Path:"alarm_status"}
      |      |       |
      |      |<------+              Code: 2.04 (Changed)
      |      | 2.04  |             Token: 0x7b
      |      |       | Object-Security: -
      |      |       |         Payload: {Code:2.05, "OFF"}
      |      |       |
      |<------+       |              Code: 2.04 (Changed)
      | 2.04 |       |             Token: 0x8c
      |      |       | Object-Security: -
      |      |       |         Payload: {Code:2.05, "OFF"}
      |      |       |
```
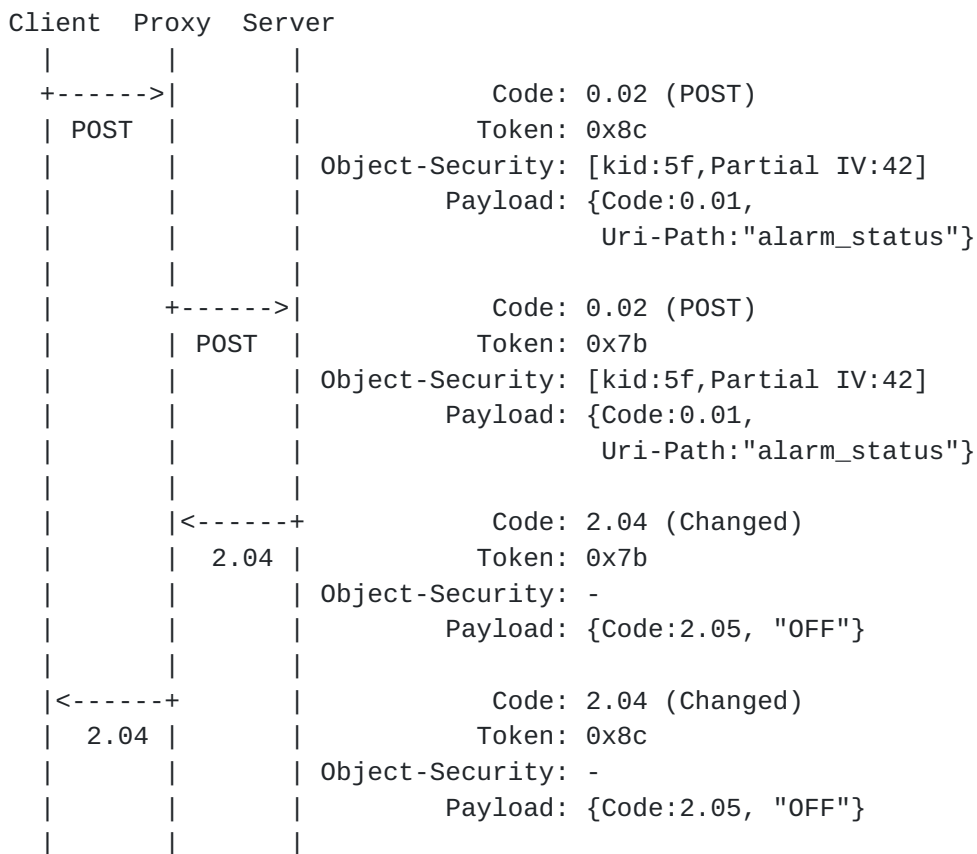
   Figure 12: Secure Access to Sensor.  Square brackets [ ... ] indicate
    content of compressed COSE object.  Curly brackets { ... } indicate
                            encrypted data.

   The request/response Codes are encrypted by OSCORE and only dummy
   Codes (POST/Changed) are visible in the header of the OSCORE message.
   The option Uri-Path ("alarm_status") and payload ("OFF") are
   encrypted.

   The COSE header of the request contains an identifier (5f),
   indicating which security context was used to protect the message and
   a Partial IV (42).

   The server verifies that the Partial IV has not been received before.
   The client verifies that the response is bound to the request.

## A.2.  Secure Subscribe to Sensor

   This example illustrates a client requesting subscription to a blood
   sugar measurement resource (GET /glucose), first receiving the value
   220 mg/dl and then a second value 180 mg/dl.

```
     Client  Proxy  Server
```

```
         |       |       |
      +------>|       |              Code: 0.05 (FETCH)
      | FETCH |       |             Token: 0x83
         |       |       |            Observe: 0
         |       |       | Object-Security: [kid:ca,Partial IV:15]
         |       |       |            Payload: {Code:0.01,
         |       |       |                    Uri-Path:"glucose"}
         |       |       |
         |     +------>|              Code: 0.05 (FETCH)
         |     | FETCH |             Token: 0xbe
         |       |       |            Observe: 0
         |       |       | Object-Security: [kid:ca,Partial IV:15]
         |       |       |            Payload: {Code:0.01,
         |       |       |                    Uri-Path:"glucose"}
         |       |       |
         |     |<------+              Code: 2.04 (Changed)
         |     | 2.04 |             Token: 0xbe
         |       |       |            Observe: 7
         |       |       | Object-Security: [Partial IV:32]
         |       |       |            Payload: {Code:2.05,
         |       |       |                    Content-Format:0, "220"}
         |       |       |
       |<------+       |              Code: 2.04 (Changed)
       | 2.04 |       |             Token: 0x83
         |       |       |            Observe: 7
         |       |       | Object-Security: [Partial IV:32]
         |       |       |            Payload: {Code:2.05,
         |       |       |                    Content-Format:0, "220"}
       ...     ...     ...
         |       |       |
         |     |<------+              Code: 2.04 (Changed)
         |     | 2.04 |             Token: 0xbe
         |       |       |            Observe: 8
         |       |       | Object-Security: [Partial IV:36]
         |       |       |            Payload: {Code:2.05,
         |       |       |                    Content-Format:0, "180"}
         |       |       |
       |<------+       |              Code: 2.04 (Changed)
       | 2.04 |       |             Token: 0x83
         |       |       |            Observe: 8
         |       |       | Object-Security: [Partial IV:36]
         |       |       |            Payload: {Code:2.05,
         |       |       |                    Content-Format:0, "180"}
         |       |       |
```

Figure 13: Secure Subscribe to Sensor.  Square brackets [ ... ]
 indicate content of compressed COSE object header.  Curly brackets {
                  ... } indicate encrypted data.

The request/response Codes are encrypted by OSCORE and only dummy
Codes (FETCH/Changed) are visible in the header of the OSCORE
message.  The options Content-Format (0) and the payload ("220" and
"180"), are encrypted.

The COSE header of the request contains an identifier (ca),
indicating the security context used to protect the message and a
Partial IV (15).  The COSE headers of the responses contains Partial
IVs (32 and 36).

The server verifies that the Partial IV has not been received before.
The client verifies that the responses are bound to the request and
that the Partial IVs are greater than any Partial IV previously
received in a response bound to the request.

## Appendix B.  Deployment examples

OSCORE may be deployed in a variety of settings, a few examples are
given in this section.

### B.1.  Master Secret Used Once

For settings where the Master Secret is only used during deployment,
the uniqueness of AEAD nonce may be assured by persistent storage of
the security context as described in this specification (see
Section 7.5).  For many IoT deployments, a 128 bit uniformly random
Master Key is sufficient for encrypting all data exchanged with the
IoT device throughout its lifetime.

### B.2.  Master Secret Used Multiple Times

In cases where the Master Secret is used to derive security context
multiple times, e.g. during recommissioning or where the security
context is not persistently stored, the reuse of AEAD nonce may be
prevented by providing a sufficiently long uniformly random byte
string as Master Salt, such that the probability of Master Salt re-
use is negligible.  The Master Salt may be transported in the Kid
Context parameter of the request (see Section 5.1)

### B.3.  Client Aliveness

The use of a single OSCORE request and response enables the client to
verify that the server's identity and aliveness through actual
communications.  While a verified OSCORE request enables the server
to verify the identity of the entity who generated the message, it
does not verify that the client is currently involved in the
communication, since the message may be a delayed delivery of a
previously generated request which now reaches the server.  To verify

the aliveness of the client the server may initiate an OSCORE
protected message exchange with the client, e.g. by switching the
roles of client and server as described in [Section 3.1](#), or by using
the Echo option in the response to a request from the client
[[I-D.ietf-core-echo-request-tag](#)].

## [Appendix C](#).  Test Vectors

This appendix includes the test vectors for different examples of
CoAP messages using OSCORE.

### [C.1](#).  Test Vector 1: Key Derivation with Master Salt

Given a set of inputs, OSCORE defines how to set up the Security
Context in both the client and the server.  The default values are
used for AEAD Algorithm and KDF.

#### [C.1.1](#).  Client

Inputs:

o  Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

o  Master Salt: 0x9e7ca92223786340 (8 bytes)

o  Sender ID: 0x (0 byte)

o  Recipient ID: 0x01 (1 byte)

From the previous parameters,

o  info (for Sender Key): 0x84400A634b657910 (8 bytes)

o  info (for Recipient Key): 0x8441010A634b657910 (9 bytes)

o  info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

o  Sender Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)

o  Recipient Key: 0xe534a26a64aa3982e988e31f1e401e65 (16 bytes)

o  Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

### C.1.2.  Server

   Inputs:

   o  Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

   o  Master Salt: 0x9e7ca92223786340 (64 bytes)

   o  Sender ID: 0x01 (1 byte)

   o  Recipient ID: 0x (0 byte)

   From the previous parameters,

   o  info (for Sender Key): 0x8441010A634b657910 (9 bytes)

   o  info (for Recipient Key): 0x84400A634b657910 (8 bytes)

   o  info (for Common IV): 0x84400a6249560d (7 bytes)

   Outputs:

   o  Sender Key: 0xe534a26a64aa3982e988e31f1e401e65 (16 bytes)

   o  Recipient Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)

   o  Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

### C.2.  Test Vector 2: Key Derivation without Master Salt

   Given a set of inputs, OSCORE defines how to set up the Security
   Context in both the client and the server.  The default values are
   used for AEAD Algorithm, KDF, and Master Salt.

### C.2.1.  Client

   Inputs:

   o  Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

   o  Sender ID: 0x00 (1 byte)

   o  Recipient ID: 0x01 (1 byte)

   From the previous parameters,

   o  info (for Sender Key): 0x8441000A634b657910 (9 bytes)

o  info (for Recipient Key): 0x8441010A634b657910 (9 bytes)

o  info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

o  Sender Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)

o  Recipient Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

o  Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

## C.2.2.  Server

Inputs:

o  Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

o  Sender ID: 0x01 (1 byte)

o  Recipient ID: 0x00 (1 byte)

From the previous parameters,

o  info (for Sender Key): 0x8441010A634b657910 (9 bytes)

o  info (for Recipient Key): 0x8441000A634b657910 (9 bytes)

o  info (for Common IV): 0x84400a6249560d (7 bytes)

Outputs:

o  Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

o  Recipient Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)

o  Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

## C.3.  Test Vector 3: OSCORE Request, Client

This section contains a test vector for a OSCORE protected CoAP GET
request using the security context derived in Appendix C.1.  The
unprotected request only contains the Uri-Path option.

Unprotected CoAP request:
0x440149c60000f2a7396c6f63616c686f737483747631 (22 bytes)

Common Context:

o  AEAD Algorithm: 10 (AES-CCM-16-64-128)

o  Key Derivation Function: HKDF SHA-256

o  Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

Sender Context:

o  Sender ID: 0x00 (1 byte)

o  Sender Key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)

o  Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

o  Partial IV: 0x14 (1 byte)

o  kid: 0x00 (1 byte)

o  external_aad: 0x8501810a4100411440 (9 bytes)

o  AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)

o  plaintext: 0x01b3747631 (5 bytes)

o  encryption key: 0xf8f3b887436285ed5a66f6026ac2cdc1 (16 bytes)

o  nonce: 0xd0a1949aa253278f34c528d2d8 (13 bytes)

From the previous parameter, the following is derived:

o  Object-Security value: 0x091400 (3 bytes)

o  ciphertext: 0x55b3710d47c611cd3924838a44 (13 bytes)

From there:

o  Protected CoAP request (OSCORE message): 0x44026dd30000acc5396c6f6
   3616c686f7374d305091400ff55b3710d47c611cd3924838a44 (37 bytes)

## C.4.  Test Vector 4: OSCORE Request, Client

This section contains a test vector for a OSCORE protected CoAP GET
request using the security context derived in Appendix C.2.  The
unprotected request only contains the Uri-Path option.

Unprotected CoAP request:
0x440149c60000f2a7396c6f63616c686f737483747631 (22 bytes)

Common Context:

o  AEAD Algorithm: 10 (AES-CCM-16-64-128)

o  Key Derivation Function: HKDF SHA-256

o  Common IV: 0x01727733ab49ead385b18f7d91 (13 bytes)

Sender Context:

o  Sender ID: 0x (0 bytes)

o  Sender Key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)

o  Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

o  Partial IV: 0x14 (1 byte)

o  kid: 0x (0 byte)

o  external_aad: 0x8501810a40411440 (8 bytes)

o  AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)

o  plaintext: 0x01b3747631 (5 bytes)

o  encryption key: 0x7230aab3b549d94c9224aacc744e93ab (16 bytes)

o  nonce: 0x01727733ab49ead385b18f7d85 (13 bytes)

From the previous parameter, the following is derived:

o  Object-Security value: 0x0914 (2 bytes)

o  ciphertext: 0x6be9214aad448260ff1be1f594 (13 bytes)

From there:

o  Protected CoAP request (OSCORE message): 0x44023bfc000066ef396c6f6
   3616c686f7374d2050914ff6be9214aad448260ff1be1f594 (36 bytes)

   This section contains a test vector for a OSCORE protected 2.05
   Content response to the request in Appendix C.3.  The unprotected
   response has payload "Hello World!" and no options.  The protected
   response does not contain a kid nor a Partial IV.

   Unprotected CoAP response:
   0x644549c60000f2a7ff48656c6c6f20576f726c6421 (21 bytes)

   Common Context:

   o  AEAD Algorithm: 10 (AES-CCM-16-64-128)

   o  Key Derivation Function: HKDF SHA-256

   o  Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

   Sender Context:

   o  Sender ID: 0x01 (1 byte)

   o  Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

   o  Sender Sequence Number: 0

   The following COSE and cryptographic parameters are derived:

   o  external_aad: 0x8501810a4100411440 (9 bytes)

   o  AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)

   o  plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)

   o  encryption key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

   o  nonce: 0xd0a1949aa253278f34c528d2d8 (13 bytes)

   From the previous parameter, the following is derived:

   o  Object-Security value: 0x (0 bytes)

   o  ciphertext: e4e8c28c41c8f31ca56eec24f6c71d94eacbcdffdc6d (22
      bytes)

   From there:

o  Protected CoAP response (OSCORE message): 0x64446dd30000acc5d008ff
   e4e8c28c41c8f31ca56eec24f6c71d94eacbcdffdc6d (33 bytes)

**[C.6](). Test Vector 6: OSCORE Response with Partial IV, Server**

This section contains a test vector for a OSCORE protected 2.05
Content response to the request in [Appendix C.3]().  The unprotected
response has payload "Hello World!" and no options.  The protected
response does not contain a kid, but contains a Partial IV.

Unprotected CoAP response:
0x644549c60000f2a7ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

o  AEAD Algorithm: 10 (AES-CCM-16-64-128)

o  Key Derivation Function: HKDF SHA-256

o  Common IV: 0xd1a1949aa253278f34c528d2cc (13 bytes)

Sender Context:

o  Sender ID: 0x01 (1 byte)

o  Sender Key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

o  Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

o  Partial IV: 0x00 (1 byte)

o  external_aad: 0x8501810a4100411440 (9 bytes)

o  AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)

o  plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)

o  encryption key: 0xd904cb101f7341c3f4c56c300fa69941 (16 bytes)

o  nonce: 0xd0a1949aa253278e34c528d2cc (13 bytes)

From the previous parameter, the following is derived:

o  Object-Security value: 0x0100 (2 bytes)

   o  ciphertext: 0xa7e3ca27f221f453c0ba68c350bf652ea096b328a1bf (22
      bytes)

   From there:

   o  Protected CoAP response (OSCORE message): 0x64442b130000b29ed20801
      00ffa7e3ca27f221f453c0ba68c350bf652ea096b328a1bf (35 bytes)

Acknowledgments

Authors' Addresses

   Goeran Selander
   Ericsson AB

   Email: goran.selander@ericsson.com


   John Mattsson
   Ericsson AB

   Email: john.mattsson@ericsson.com


   Francesca Palombini
   Ericsson AB

   Email: francesca.palombini@ericsson.com


   Ludwig Seitz
   RISE SICS

   Email: ludwig.seitz@ri.se