CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2018

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

June 27, 2018

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-13

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

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer 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-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require DTLS or TLS [RFC5246] 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], Block-wise [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 protects neither 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 [RFC8323] 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 with non-IP transports (e.g., [I-D.bormann-6lo-coap-802-15-ie]). OSCORE may also be used in different ways with HTTP. OSCORE messages may be transported in HTTP, and OSCORE may also be used to protect CoAP-mappable HTTP messages, as described below.

OSCORE is designed to protect as much information as possible while still allowing CoAP proxy operations (Section 10). It works with existing 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 11. OSCORE may be used together with TLS or DTLS over one or more hops in the end-to-end path, e.g. transported with HTTPS in one hop and with plain CoAP in another hop. 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 binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled in CoAP with a new option (Section 2), and in HTTP with a new header field (Section 11.1) and content type (Section 13.5). 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 OSCORE option/header field is included in the message. A sketch of an exchange of OSCORE messages, in the case of the original message being CoAP, is provided in Figure 2.

```
       Client                                           Server
          |      OSCORE request - POST example.com:      |
          |        Header, Token,                        |
          |        Options: {OSCORE, ...},               |
          |        Payload: COSE ciphertext              |
          +--------------------------------------------->|
          |                                              |
          |<---------------------------------------------+
          |      OSCORE response - 2.04 (Changed):       |
          |        Header, Token,                        |
          |        Options: {OSCORE, ...},               |
          |        Payload: COSE ciphertext              |
          |                                              |
```

Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, 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, ID Context, and Common IV are defined in Section 3.1.

2. The OSCORE Option

The OSCORE option (see Figure 3, which extends Table 4 of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Sections 5 and 6). The OSCORE option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1	x				OSCORE	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
(*) See below.

Figure 3: The OSCORE Option

The OSCORE option includes the OSCORE flag bits (Section 6), the Sender Sequence Number, the Sender ID, and the ID Context when these fields are present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits are all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an OSCORE option SHALL treat it as malformed and reject it.

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

For CoAP proxy operations, see Section 10.

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, [RFC5116]) 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:

- AEAD Algorithm. The COSE AEAD algorithm to use for encryption.

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

- Master Secret. Variable length, random byte string (see Section 12.3) used to derive traffic keys and IVs.

- Master Salt. Optional variable length byte string containing the salt used to derive traffic keys and IVs.

- ID Context. Optional variable length byte string providing additional information to identify the Common Context and to derive traffic keys and IVs.

- Common IV. Byte string derived from Master Secret, Master Salt, and ID Context. Length is determined by the AEAD Algorithm.

The Sender Context contains the following parameters:

- Sender ID. Byte string used to identify the Sender Context, to derive traffic keys and IVs, and to assure unique nonces. Maximum length is determined by the AEAD Algorithm.

- 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.

- Sender Sequence Number. Non-negative integer used by the sender to protect requests and certain responses, e.g. 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:

- Recipient ID. Byte string used to identify the Recipient Context, to derive traffic keys and IVs, and to assure unique nonces. Maximum length is determined by the AEAD Algorithm.

- 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.

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

All parameters except Sender Sequence Number and Replay Window are immutable once the security context is established. An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them 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:

- Master Secret

- Sender ID

- 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:

- AEAD Algorithm

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

- Master Salt

  * Default is the empty byte string

- Key Derivation Function (KDF)

  * Default is HKDF SHA-256

- 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. The way the input parameters are pre-established, is application specific. Considerations of security context establishment are given in Section 12.2 and examples of deploying OSCORE 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 for COSE [RFC8152]. 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:

- salt is the Master Salt as defined above

- IKM is the Master Secret as defined above

- info is the serialization of a CBOR array consisting of:

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

where:

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

- id_context is the ID Context, or nil if ID Context is not provided.

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

- 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.

Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros. For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string. OSCORE sets the salt default value to empty byte string, which in [RFC5869] is converted to a string of zeroes (see Section 2.2 of [RFC5869]).

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

To ensure unique Sender Keys, the quartet (Master Secret, Master Salt, ID Context, Sender ID) MUST be unique, i.e. the pair (ID Context, Sender ID) SHALL be unique in the set of all security contexts using the same Master Secret and Master Salt. The requirement that Sender ID SHALL be unique in the set of all security contexts using the same Master Secret, Master Salt, and ID Context guarantees unique (key, nonce) pairs, which avoids nonce reuse.

Different methods can be used to assign Sender IDs: a protocol that allows the parties to negotiate locally unique identifiers, a trusted third party (e.g., [I-D.ietf-ace-oauth-authz]), or the identifiers can be assigned out-of-band. The Sender IDs can be very short (note that the empty string is a legitimate value). 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.

To simplify retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint. If an endpoint has the same Recipient ID with different Recipient Contexts, i.e. the Recipient Contexts are derived from different Common Contexts, then the endpoint may need to try multiple times before verifying the right security context associated to the Recipient ID.

The ID Context is used to distinguish between security contexts. The methods used for assigning Sender ID can also be used for assigning the ID Context. Additionally, the ID Context can be generated by the client (see Appendix B.2). ID Context can be arbitrarily long.

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 Sections 10 and 11). 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 focus on 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 11. 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 [RFC8323]) is specified in Section 4.3. 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:

- Class E: encrypted and integrity protected,

- Class I: integrity protected only, or

- 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 enable proxy operations.

4.1. CoAP Options

A summary of how options are protected is shown in Figure 5. Note that some options may have both Inner and Outer message fields which are protected accordingly. Certain options require special processing as is described in Section 4.1.3.

No.	Name	E	U
1	If-Match	x	
3	Uri-Host		x
4	ETag	x	
5	If-None-Match	x	
6	Observe	x	x
7	Uri-Port		x
8	Location-Path	x	
TBD1	OSCORE		x
11	Uri-Path	x	
12	Content-Format	x	
14	Max-Age	x	x
15	Uri-Query	x	
17	Accept	x	
20	Location-Query	x	
23	Block2	x	x
27	Block1	x	x
28	Size2	x	x
35	Proxy-Uri		x
39	Proxy-Scheme		x
60	Size1	x	x
258	No-Response	x	x

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

Figure 5: 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. If a new CoAP option is of class U, the potential issues with the option being unprotected SHOULD be documented (see Appendix D.4).

4.1.1. 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 Sections 8.2 and 8.4.

4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations, see Appendix D.1.

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 the OSCORE option, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Outer option message field.

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

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.4. Specifications that introduce repeatable Class I options MUST specify that proxies MUST NOT change the order of the instances of such an option in the CoAP message.

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

4.1.3. Special Options

Some options require special processing as specified in this section.

4.1.3.1. 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 a normal Inner option, specified in Section 4.1.1.

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, which are described in Sections 7.4, 8.2, and 8.4. Such a message field is then processed according to Section 4.1.2.

Successful OSCORE responses do not need to include an Outer Max-Age option since the responses are non-cacheable by construction (see Section 4.2).

4.1.3.2. Uri-Host and Uri-Port

When the Uri-Host and Uri-Port are set to their default values (see Section 5.10.1 [RFC7252]), they are omitted from the message (Section 5.4.4 of [RFC7252]), which is favorable both for overhead and privacy.

In order to support forward proxy operations, Proxy-Scheme, Uri-Host, and Uri-Port need to be Class U. For the use of Proxy-Uri, see Section 4.1.3.3.

Manipulation of unprotected message fields (including Uri-Host, Uri-Port, destination IP/port or request scheme) MUST NOT lead to an OSCORE message becoming verified by an unintended server. Different servers SHOULD have different security contexts.

4.1.3.3. Proxy-Uri

When Proxy-Uri is present, the client 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 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 4.1.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host, and Uri-Port options as

specified in Section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.1.2).

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 (see Section 11.2). In future specifications of 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 specifying the OSCORE processing.

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 4.1.1, and are thus encrypted and transported in the COSE object:

o Uri-Path = "resource"

o Uri-Query = "q=1"

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 details.

4.1.3.4. The Block Options

Block-wise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting block-wise transfers. The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message segmentation such that each segment 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 Block-wise transfers provided all blocks are received.

4.1.3.4.1. 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 normal Inner options (Section 4.1.1). The receiving CoAP endpoint SHALL process the OSCORE message before processing Block-wise as defined in [RFC7959].

4.1.3.4.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.1.2). 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 4.1.3.4.1).

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].

4.1.3.5. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7641], in which case the Observe related processing can be omitted.

The support for Observe [RFC7641] with OSCORE targets the requirements on forwarding of Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs], i.e. that observations go through intermediary nodes, as illustrated in Figure 8 of [RFC7641].

Inner Observe SHALL be used to protect the value of the Observe option between the endpoints. Outer Observe SHALL be used to support forwarding by intermediary nodes.

The server SHALL include a new Partial IV in responses (with or without the Observe option) to Observe registrations.

[RFC7252] does not specify how the server should act upon receiving the same Token in different requests. When using OSCORE, the server SHOULD NOT remove an active observation just because it receives a request with the same Token.

Since POST with Observe is not defined, for messages with Observe, the Outer Code MUST be set to 0.05 (FETCH) for requests and to 2.05 (Content) for responses (see Section 4.2).

4.1.3.5.1. Registrations and Cancellations

The Inner and Outer Observe in the request MUST contain the Observe value of the original CoAP request; 0 (registration) or 1 (cancellation).

Every time a client issues a new Observe request, a new Partial IV MUST be used (see Section 5), and so the payload and OSCORE option are changed. The server uses the Partial IV of the new request as the 'request_piv' of all associated notifications (see Section 5.4). The Partial IV of the registration is also used as 'request_piv' of associated cancellations (see Section 5.4).

Intermediaries are not assumed to have access to the OSCORE security context used by the endpoints, and thus cannot make requests or transform responses with the OSCORE option which verify at the

receiving endpoint as coming from the other endpoint. This has the following consequences and limitations for Observe operations.

- An intermediary node removing the Outer Observe 0 does not change the registration request to a request without Observe (see Section 2 of [RFC7641]). Instead other means for cancellation may be used as described in Section 3.6 of [RFC7641].

- An intermediary node is not able to transform a normal response into an OSCORE protected Observe notification (see figure 7 of [RFC7641]) which verifies as coming from the server.

- An intermediary node is not able to initiate an OSCORE protected Observe registration (Observe with value 0) which verifies as coming from the client. An OSCORE-aware intermediary SHALL NOT initiate registrations of observations (see Section 10). If an OSCORE-unaware proxy re-sends an old registration message from a client this will trigger the replay protection mechanism in the server. To prevent this from resulting in the OSCORE-unaware proxy to cancel of the registration, a server MAY respond to a replayed registration request with a replay of a cached notification. Alternatively, the server MAY send a new notification.

- An intermediary node is not able to initiate an OSCORE protected Observe cancellation (Observe with value 1) which verifies as coming from the client. An application MAY decide to allow intermediaries to cancel Observe registrations, e.g. to send Observe with value 1 (see Section 3.6 of [RFC7641]), but that can also be done with other methods, e.g. reusing the Token in a different request or sending a RST message. This is out of scope for this specification.

4.1.3.5.2. Notifications

If the server accepts an Observe registration, a Partial IV MUST be included in all notifications (both successful and error). To protect against replay, 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 received notifications for the associated Observe registration. Further details of replay protection of notifications are specified in Section 7.4.1.

For notifications, the Inner Observe value MUST be empty (see Section 3.2 of [RFC7252]). The Outer Observe in a notification is needed for intermediary nodes to allow multiple responses to one request, and may be set to the value of Observe in the original CoAP

message. The client performs ordering of notifications and replay protection by comparing their Partial IVs and SHALL ignore the outer Observe value.

If the client receives a response to an Observe request without an Inner Observe option, then it verifies the response as a non-Observe response, as specified in Section 8.4. If the client receives a response to a non-Observe request with an Inner Observe option, then it stops processing the message, as specified in Section 8.4.

A client MUST consider the notification with the highest Partial IV as the freshest, regardless of the order of arrival. In order to support existing Observe implementations the OSCORE client implementation MAY set the Observe value to the three least significant bytes of the Partial IV.

4.1.3.6. No-Response

No-Response [RFC7967] is an optional feature used by the client to communicate its disinterest in certain classes of responses to a particular request. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7967].

If used, No-Response MUST be Inner. The Inner No-Response SHALL be processed by OSCORE as specified in Section 4.1.1. The Outer option SHOULD NOT be present. The server SHALL ignore the Outer No-Response option. The client MAY set the Outer No-Response value to 26 ('suppress all known codes') if the Inner value is set to 26. The client MUST be prepared to receive and discard 5.04 Gateway Timeout error messages from intermediaries potentially resulting from destination time out due to no response.

4.1.3.7. OSCORE

The OSCORE option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed. The content in the OSCORE 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). Nested use of OSCORE is not supported: If OSCORE processing detects an OSCORE option in the original CoAP message, then processing SHALL be stopped.

4.2. CoAP Header Fields and Payload

A summary of how the CoAP header fields and payload are protected is shown in Figure 6, 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
Payload	x	

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

Figure 6: Protection of CoAP Header Fields and Payload

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 on 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 sending endpoint writes an Outer Code to the OSCORE message. With one exception (see Section 4.1.3.5) the Outer Code SHALL be set to 0.02 (POST) for requests and to 2.04 (Changed) for responses. The receiving endpoint SHALL discard the Outer Code in the OSCORE message and write the Code of the COSE object plaintext (Section 5.3) into the decrypted CoAP message.

The other currently defined 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.

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

4.3. 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 [RFC8323].

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

- To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.

- 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.

- 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).

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

NOTE: Option numbers for Signaling messages are specific to the CoAP Code (see Section 5.2 of [RFC8323]).

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 12.

As specified in [RFC5116], plaintext denotes the data that is to be encrypted and integrity protected, and Additional Authenticated Data (AAD) denotes the data that is to be 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 bytes of value zero SHALL be removed when encoding the Partial IV, except in the case of Partial IV of value 0 which is encoded to the byte string 0x00. This parameter SHALL be present in requests. The Partial IV SHALL be present in responses to Observe registrations (see Section 4.1.3.5.1), otherwise the Partial IV will not typically be present in responses.

  * The 'kid' parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and will not typically 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.ietf-core-oscore-groupcomm].

  * Optionally, a 'kid context' parameter (see Section 5.1) containing an ID Context (see Section 3.1). This parameter MAY be present in requests and MUST NOT be present in responses. If 'kid context' is present in the request, then the server SHALL use a security context with that ID Context when verifying the request.

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 Sender ID is used with multiple contexts, it is possible (and sometimes necessary,

see Section 3.3) for the client to use an ID Context to distinguish the security contexts (see Section 3.1). For example:

- o If the client has a unique identifier in some namespace, then that identifier can be used as ID Context.

- o In case of group communication [I-D.ietf-core-oscore-groupcomm], a group identifier can be used as ID Context to enable different security contexts for a server belonging to multiple groups.

The Sender ID and Context ID are used to establish the necessary input parameters and in the derivation of the security context (see Section 3.2). Whereas the 'kid' parameter is used to transport the Sender ID, the new COSE header parameter 'kid context' is used to transport the ID Context, see Figure 7.

name	label	value type	value registry	description
kid context	TBD2	bstr		Identifies the context for kid

Figure 7: Common Header Parameter kid context for the COSE object

5.2. Nonce

The AEAD nonce is constructed in the following way (see Figure 8):

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

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

3. concatenating the size of the ID_PIV (a single byte S) with the padded ID_PIV and the padded PIV,

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_PIV, and PIV together with endpoint unique IDs and encryption keys makes it easy to verify that the nonces used with a specific key will be unique, see Appendix D.3.

If the Partial IV is not present in a response, the nonce from the request is used. For responses that are not notifications (i.e. when there is a single response to a request), the request and the response should typically use the same nonce to reduce message overhead. Both alternatives provide all the required security properties, see Section 7.4 and Appendix D.3. The only non-Observe scenario where a Partial IV must be included in a response is when the server is unable to perform replay protection, see Section 7.5.2. For processing instructions see Section 8.

```
          <- nonce length minus 6 B -> <-- 5 bytes -->
    +---+-------------------+--------+---------+-----+
    | S |      padding      | ID_PIV | padding | PIV |----+
    +---+-------------------+--------+---------+-----+    |
                                                          |
     <---------------- nonce length ---------------->     |
    +------------------------------------------------+    |
    |                   Common IV                    |->(XOR)
    +------------------------------------------------+    |
                                                          |
     <---------------- nonce length ---------------->     |
    +------------------------------------------------+    |
    |                     Nonce                      |<---+
    +------------------------------------------------+
```

Figure 8: AEAD Nonce Formation

5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

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

- all Inner option message fields (see Section 4.1.1) present in the original CoAP message (see Section 4.1). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Class E option; and

- 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 9: 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,
   algorithms : [ alg_aead : int / tstr ],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr
]
```

where:

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

- algorithms: contains (for extensibility) an array of algorithms, according to this specification only containing alg_aead.

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

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

- request_piv: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5), with one exception: in case of protection or verification of Observe cancellations, the request_piv contains the value of the 'Partial IV' in the COSE object of the corresponding registration (see Section 4.1.3.5.1).

o options: contains the Class I options (see Section 4.1.2) 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 instance of class I option.

The oscore_version and algorithms parameters are established out-of-band and are thus never transported in OSCORE, but the external_aad allows to verify that they are the same in both endpoints.

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 all requests.

The Additional Authenticated Data (AAD) is composed from the external_add as described in Section 5.3 of [RFC8152].

6. OSCORE Header 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 header 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".

The COSE_Encrypt0 object used in OSCORE is transported in the OSCORE option and in the Payload. The Payload contains the Ciphertext of the COSE object. The headers of the COSE object are compactly encoded as described in the next section.

6.1. Encoding of the OSCORE Option Value

The value of the OSCORE 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 10: The OSCORE Option Value

- 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.

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

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

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

- 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 OSCORE option value, even in case reserved bits are used and additional fields are added to it.

The length of the OSCORE 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

This section covers a list of OSCORE Header Compression examples for requests and responses. The examples assume the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the full CoAP unprotected message, as well as the full security context, is not reported in the examples, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 6, divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, and Partial IV = 0x05

   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)
   ```

2. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, and Partial IV = 0x00

Before compression (23 bytes):

```
[
h'',
{ 4:h'', 6:h'00' },
h'aea0155667924dff8a24e4cb35b9'
]
```

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)

3. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, Partial IV = 0x05, and kid context = 0x44616c656b

Before compression (30 bytes):

```
[
h'',
{ 4:h'', 6:h'05', 8:h'44616c656b' },
h'aea0155667924dff8a24e4cb35b9'
]
```

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)

4. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and no Partial IV

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)

5. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and Partial IV = 0x07

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. Message Binding, Sequence Numbers, Freshness and Replay Protection

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 intermediaries, 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. Sequence Numbers

An AEAD nonce MUST NOT be used more than once per AEAD key. The uniqueness of (key, nonce) pairs is shown in Appendix D.3, and in particular depends on a correct usage of Partial IVs. If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

7.2.1. Maximum Sequence Number

The maximum Sender Sequence Number is algorithm dependent (see Section 12), and SHALL be less than 2^40. If the Sender Sequence Number exceeds the maximum, the endpoint MUST NOT process any more messages with the given Sender Context. If necessary, the endpoint SHOULD acquire a new security context before this happens. The latter is out of scope of this document.

7.3. Freshness

For requests, OSCORE provides only the guarantee 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].

Assuming an honest server (see Appendix D), the message binding guarantees that a response is not older than its request. For responses that are not notifications (i.e. when there is a single response to a request), this gives absolute freshness. For notifications, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly re-register the observation. Note that the message binding does not guarantee that misbehaving server created the response before receiving the request, i.e. it does not verify server aliveness.

For requests and notifications, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of requests or 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, to inform any intermediary that the response is not to be cached. The diagnostic payload MAY contain the "Replay detected" 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 (with or without Partial IV) are protected against replay as they are bound to the request and the fact that only a single response is accepted. Note that the Partial IV is not used for replay protection in this case.

The operation of validating the Partial IV and updating the replay protection MUST be atomic.

7.4.1. Replay Protection of Notifications

The following applies additionally when Observe is supported.

The Notification Number is initialized to the Partial IV of the first successfully verified notification in response to the registration request. A client receiving a notification SHALL compare the Partial IV with the Notification Number associated to that Observe registration. The client MUST stop processing notifications with a Partial IV which has been previously received. Applications MAY decide that a client only processes notifications which have greater Partial IV than the Notification Number.

If the verification of the response succeeds, and the received Partial IV was greater than the Notification Number then the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

7.5. Losing Part of the Context State

To prevent reuse of an 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 can either use a persistently stored complete or partial security context, or establish a new security context with each endpoint it communicates with. However, establishing a fresh security context may have a non-negligible cost in terms of, e.g., power consumption.

If the endpoint uses a persistently stored partial security context, it MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously received messages. Some ways to achieve this are described in the following sections.

7.5.1. Sequence Number

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

- o Before using a Sender Sequence Number that 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. 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 of Notifications

To prevent accepting replay of previously received notifications, the client may perform the following procedure after boot:

- o The client forgets about earlier registrations, removes all Notification Numbers and registers using Observe.

8. Processing

This section describes the OSCORE message processing. Additional processing for Observe or Block-wise are described in subsections.

Note that, analogously to [RFC7252] where the Token and source/destination pair are used to match a response with a request, both endpoints MUST keep the association (Token, {Security Context, Partial IV of the request}), in order to be able to find the Security Context and compute the AAD to protect or verify the response. The association MAY be forgotten after it has been used to successfully

protect or verify the response, with the exception of Observe processing, where the association MUST be kept as long as the Observation is active.

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 Sections 5.3 and 5.4.

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

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 OSCORE option is added (see Section 4.1.2).

8.2. Verifying the Request

A server receiving a request containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

2. Decompress the COSE Object (Section 6) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter, additionally using the 'kid context', if present. 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".

* If 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".

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

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

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

6. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)

   * 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 MAY contain the "Decryption failed" string.

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

7. Add decrypted Code, options, and payload to the decrypted request. The OSCORE option is removed.

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

8.2.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

A. If Block-wise is present in the request then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

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 (step 8 in Section 8.2) are protected, as well as successful CoAP responses, while the OSCORE errors (steps 2, 3, and 6 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 associated with the Token.

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

3. Compute the AEAD nonce as described in Section 5.2:

   * Either use the nonce from the request, or

   * Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.

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 OSCORE option is added (see Section 4.1.2).

8.3.1. Supporting Observe

If Observe is supported, insert the following step between step 2 and 3 of Section 8.3:

A. If the request was a registration, encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV, then go to 4.

8.4. Verifying the Response

A client receiving a response containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, ETag Outer option is discarded, as well as Max-Age Outer option.

2. Retrieve the Recipient Context in the Security 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 8.

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

4. Compute the AEAD nonce

   * If the Partial IV is not present in the response, the nonce from the request is used.

   * 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.

5. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.) If decryption fails, then go to 8.

6. Add decrypted Code, options and payload to the decrypted request. The OSCORE option is removed.

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

8. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response.

8.4.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

A. If Block-wise is present in the request then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

8.4.2. Supporting Observe

If Observe is supported:

Insert the following step between step 5 and step 6:

A. If the request was an Observe registration, then:

o If the Partial IV is not present in the response, and either the client has previously received a successful notification to the

registration (active observation) or Inner Observe is present, then go to 8.

- If the Partial IV is present in the response and Inner Observe is present, then follow the processing described in Section 4.1.3.5.2 and Section 7.4.1, then:

  * initialize the Notification Number (if first successfully verified notification), or

  * overwrite the Notification Number (if the received Partial IV was greater than the Notification Number).

Replace step 8 of Section 8.4 with:

B. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response. An error condition occurring while processing a response to an observation request does not cancel the observation. A client MUST NOT react to failure 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, e.g. using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "osc" attribute is a hint indicating that the destination of that link is only accessible using OSCORE, and unprotected access to it is not supported. 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.

The example in Figure 11 shows a use of the "osc" attribute: the client does resource discovery on a server, and gets back a list of resources, one of which includes the "osc" attribute indicating that the resource is protected with OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core

RES: 2.05 Content
   </sensors/temp>;osc,
   </sensors/light>;if="sensor"
```

Figure 11: The web link

10. CoAP-to-CoAP Forwarding Proxy

CoAP is designed for proxy operations (see Section 5.7 of [RFC7252]).

OSCORE is designed to work with OSCORE-unaware CoAP proxies. Security requirements for forwarding are listed in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. Proxy processing of the (Outer) Proxy-Uri option works as defined in [RFC7252]. Proxy processing of the (Outer) Block options works as defined in [RFC7959].

However, not all CoAP proxy operations are useful:

- Since a CoAP response is only applicable to the original CoAP request, caching is in general not useful. In support of existing proxies, OSCORE uses the outer Max-Age option, see Section 4.1.3.1.

- Proxy processing of the (Outer) Observe option as defined in [RFC7641] is specified in Section 4.1.3.5.

Optionally, a CoAP proxy MAY detect OSCORE and act accordingly. An OSCORE-aware CoAP proxy:

- SHALL bypass caching for the request if the OSCORE option is present

- SHOULD avoid caching responses to requests with an OSCORE option

In the case of Observe (see Section 4.1.3.5) the OSCORE-aware CoAP proxy:

- SHALL NOT initiate an Observe registration

- MAY verify the order of notifications using Partial IV rather than the Observe option

11. HTTP Operations

The CoAP request/response model may be mapped to HTTP and vice versa as described in Section 10 of [RFC7252]. The HTTP-CoAP mapping is further detailed in [RFC8075]. This section defines the components needed to map and transport OSCORE messages over HTTP hops. By mapping between HTTP and CoAP and by using cross-protocol proxies OSCORE may be used end-to-end between e.g. an HTTP client and a CoAP server. Examples are provided at the end of the section.

11.1. The HTTP OSCORE Header Field

The HTTP OSCORE Header Field (see Section 13.4) is used for carrying the content of the CoAP OSCORE option when transporting OSCORE messages over HTTP hops.

The HTTP OSCORE header field is only used in POST requests and 200 (OK) responses. When used, the HTTP header field Content-Type is set to 'application/oscore' (see Section 13.5) indicating that the HTTP body of this message contains the OSCORE payload (see Section 6.2). No additional semantics is provided by other message fields.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP OSCORE header field value is as follows.

```
base64url-char = ALPHA / DIGIT / "-" / "_"

OSCORE = 2*base64url-char
```

The HTTP OSCORE header field is not appropriate to list in the Connection header field (see Section 6.1 of [RFC7230]) since it is not hop-by-hop. OSCORE messages are generally not useful when served from cache (i.e., they will generally be marked Cache-Control: no-cache) and so interaction with Vary is not relevant (Section 7.1.4 of [RFC7231]). Since the HTTP OSCORE header field is critical for message processing, moving it from headers to trailers renders the message unusable in case trailers are ignored (see Section 4.1 of [RFC7230]).

Intermediaries are in general not allowed to insert, delete, or modify the OSCORE header. Changes to the HTTP OSCORE header field will in general violate the integrity of the OSCORE message resulting in an error. For the same reason the HTTP OSCORE header field is in general not preserved across redirects.

Since redirects are not defined in the mappings between HTTP and CoAP [RFC8075][RFC7252], a number of conditions need to be fulfilled for redirects to work. For CoAP client to HTTP server, such conditions include:

- the CoAP-to-HTTP proxy follows the redirect, instead of the CoAP client as in the HTTP case

- the CoAP-to-HTTP proxy copies the HTTP OSCORE header field and body to the new request

- the target of the redirect has the necessary OSCORE security context required to decrypt and verify the message

Since OSCORE requires HTTP body to be preserved across redirects, the HTTP server is recommended to reply with 307 or 308 instead of 301 or 302.

For the case of HTTP client to CoAP server, although redirect is not defined for CoAP servers [RFC7252], an HTTP client receiving a redirect should generate a new OSCORE request for the server it was redirected to.

11.2. CoAP-to-HTTP Mapping

Section 10.1 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP cross-protocol mapping process. The additional rules for OSCORE messages are:

- The HTTP OSCORE header field value is set to

  * AA if the CoAP OSCORE option is empty, otherwise

  * the value of the CoAP OSCORE option (Section 6.1) in base64url (Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

- The HTTP Content-Type is set to 'application/oscore' (see Section 13.5), independent of CoAP Content-Format.

11.3. HTTP-to-CoAP Mapping

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. The additional rules for HTTP messages with the OSCORE header field are:

- The CoAP OSCORE option is set as follows:

   * empty if the value of the HTTP OSCORE header field is a single zero byte (0x00) represented by AA, otherwise

   * the value of the HTTP OSCORE header field decoded from base64url (Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].

o The CoAP Content-Format option is omitted, the content format for OSCORE (Section 13.6) MUST NOT be used.

11.4. HTTP Endpoints

Restricted to subsets of HTTP and CoAP supporting a bijective mapping, OSCORE can be originated or terminated in HTTP endpoints.

The sending HTTP endpoint uses [RFC8075] to translate the HTTP message into a CoAP message. The CoAP message is then processed with OSCORE as defined in this document. The OSCORE message is then mapped to HTTP as described in Section 11.2 and sent in compliance with the rules in Section 11.1.

The receiving HTTP endpoint maps the HTTP message to a CoAP message using [RFC8075] and Section 11.3. The resulting OSCORE message is processed as defined in this document. If successful, the plaintext CoAP message is translated to HTTP for normal processing in the endpoint.

11.5. Example: HTTP Client and CoAP Server

This section is giving an example of how a request and a response between an HTTP client and a CoAP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps.

Mapping and notation here is based on "Simple Form" (Section 5.4.1 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
  Content-Type: application/oscore
  OSCORE: CSU
  Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]

[CoAP request -- Proxy to CoAP Server]

  POST coap://server.url/
  OSCORE: 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
  OSCORE: [empty]
  Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[HTTP response -- Proxy to HTTP Client]

  HTTP/1.1 200 OK
  Content-Type: application/oscore
  OSCORE: AA
  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.

11.6. Example: CoAP Client and HTTP Server

This section is giving an example of how a request and a response between a CoAP client and an HTTP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps

```
[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/
  OSCORE: 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
  Content-Type: application/oscore
  OSCORE: CSU
  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
  Content-Type: application/oscore
  OSCORE: AA
  Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- Proxy to CoAP Client]

  2.04 Changed
  OSCORE: [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.

12. Security Considerations

An overview of the security properties is given in Appendix D.

12.1. 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 any 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. In particular when OSCORE is used with HTTP, the additional TLS protection of HTTP hops is recommended, e.g. between an HTTP endpoint and a proxy translating between HTTP and CoAP.

Applications need to consider that certain message fields and messages types are not protected end-to-end and may be spoofed or manipulated. The consequences of unprotected message fields are analyzed in Appendix D.4.

12.2. Security Context Establishment

The use of COSE_Encrypt0 and AEAD 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 Master Secret and unique Sender IDs. The necessary input parameters may be pre-established or obtained using a key establishment protocol augmented with establishment of Sender/Recipient ID such as the OSCORE profile of the ACE framework [I-D.ietf-ace-oscore-profile]. Such a procedure must ensure that the requirements of the security context parameters for the intended use are complied with (see Section 3.3) and also in error situations. It is recommended to use a key establishment protocol which provides forward secrecy whenever possible. Considerations for deploying OSCORE with a fixed Master Secret are given in Appendix B.

12.3. Master Secret

OSCORE uses HKDF [RFC5869] and the established input parameters to derive the security context. The required properties of the security context parameters are discussed in Section 3.3, in this section we focus on the Master Secret. HKDF denotes in this specification the composition of the expand and extract functions as defined in [RFC5869] and the Master Secret is used as Input Key Material (IKM).

Informally, HKDF takes as source an IKM containing some good amount of randomness but not necessarily distributed uniformly (or for which an attacker has some partial knowledge) and derive from it one or more cryptographically strong secret keys [RFC5869].

Therefore, the main requirement for the OSCORE Master Secret, in addition to being secret, is that it is has a good amount of randomness. The selected key establishment schemes must ensure that the necessary properties for the Master Secret are fulfilled. For pre-shared key deployments and key transport solutions such as [I-D.ietf-ace-oscore-profile], the Master Secret can be generated offline using a good random number generator.

12.4. Replay Protection

Replay attacks need to be considered in different parts of the implementation. 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. An adversary may

try to induce a device reboot for the purpose of replaying a message (see Section 7.5).

Note that sharing a security context between servers may open up for replay attacks, for example if the replay windows are not synchronized.

12.5. Client Aliveness

A verified OSCORE request enables the server to verify the identity of the entity who generated the message. However, 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 use the Echo option in the response to a request from the client (see [I-D.ietf-core-echo-request-tag]).

12.6. Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number is 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 bits, and the Partial IV is at most 40 bits. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*.

In order to prevent cryptanalysis when the same plaintext is repeatedly encrypted by many different users with distinct keys, the nonce is formed by mixing the sequence number with a secret per-context initialization vector (Common IV) derived along with the keys (see Section 3.1 of [RFC8152]), and by using a Master Salt in the key derivation (see [MF00] for an overview). The Master Secret, Sender Key, Recipient Key, and Common IV must be secret, the rest of the parameters may be public. The Master Secret must have a good amount of randomness (see Section 12.3).

12.7. Message Segmentation

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.

12.8. 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 5) may reveal privacy sensitive information, see Appendix D.4. 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. OSCORE does not provide protection for HTTP header fields which are not both CoAP-mappable and class E. The HTTP message fields which are visible to on-path entity are only used for the purpose of transporting the OSCORE message, whereas the application layer message is encoded in CoAP and encrypted.

COSE message fields, i.e. the OSCORE option, may reveal information about the communicating endpoints. E.g. 'kid' and 'kid context', which are intended to help the server find the right context, may reveal information about the client. Tracking 'kid' and 'kid context' to one server may be used for correlating requests from one client.

Unprotected error messages reveal information about the security state in the communication between the endpoints. Unprotected signaling 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.

13. 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 "TBDx" in this specification should be assigned the same number.

13.1. COSE Header Parameters Registry

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

- o Name: kid context
- o Label: TBD2
- o Value Type: bstr
- o Value Registry:
- o Description: Identifies the context for kid
- o Reference: Section 5.1 of this document

Note to IANA: Label assignment in (Integer value between 1 and 255) is requested. (RFC Editor: Delete this note after IANA assignment)

13.2. CoAP Option Numbers Registry

The OSCORE option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD1	OSCORE	[[this document]]

13.3. CoAP Signaling Option Numbers Registry

The OSCORE option is added to the CoAP Signaling Option Numbers registry:

Applies to	Number	Name	Reference
7.xx (any)	TBD1	OSCORE	[[this document]]

13.4. Header Field Registrations

The HTTP OSCORE header field is added to the Message Headers registry:

Header Field Name	Protocol	Status	Reference
OSCORE	http	standard	[[this document]]

13.5. Media Type Registrations

This section registers the 'application/oscore' media type in the "Media Types" registry. These media types are used to indicate that the content is an OSCORE message. The OSCORE body cannot be understood without the OSCORE header field value and the security context.

Type name: application

Subtype name: oscore

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[This document]].

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: IoT applications sending security content over HTTP(S) transports.

Fragment identifier considerations: N/A

Additional information:

* Deprecated alias names for this type: N/A

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

13.6. CoAP Content-Formats Registry

Note to IANA: ID assignment in the 10000-64999 range is requested. (RFC Editor: Delete this note after IANA assignment)

This section registers the media type 'application/oscore' media type in the "CoAP Content-Format" registry. This Content-Format for the OSCORE payload is defined for potential future use cases and SHALL NOT be used in the OSCORE message. The OSCORE payload cannot be understood without the OSCORE option value and the security context.

Media Type	Encoding	ID	Reference
application/oscore		TBD3	[[this document]]

14. References

14.1. 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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

14.2. 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-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-12 (work in progress), May 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-01 (work in progress), March 2018.

[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-02 (work in progress), February 2018.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-01 (work in progress), March 2018.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-01 (work in progress), March 2018.

[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-05 (work in progress), March 2018.

[MF00] McGrew, D. and S. Fluhrer, "Attacks on Encryption of Redundant Plaintext and Implications on Internet Security", the Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag. , 2000.

[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>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/info/rfc6690>.

[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
  |       |       |          OSCORE: [kid:5f,Partial IV:42]
  |       |       |         Payload: {Code:0.01,
  |       |       |                   Uri-Path:"alarm_status"}
  |       |       |
  |       +------>|            Code: 0.02 (POST)
  |       | POST  |           Token: 0x7b
  |       |       |          OSCORE: [kid:5f,Partial IV:42]
  |       |       |         Payload: {Code:0.01,
  |       |       |                   Uri-Path:"alarm_status"}
  |       |       |
  |       |<------+            Code: 2.04 (Changed)
  |       |  2.04 |           Token: 0x7b
  |       |       |          OSCORE: -
  |       |       |         Payload: {Code:2.05, "0"}
  |       |       |
  |<------+       |            Code: 2.04 (Changed)
  |  2.04 |       |           Token: 0x8c
  |       |       |          OSCORE: -
  |       |       |         Payload: {Code:2.05, "0"}
  |       |       |
```

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 ("0") 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 the request as specified in Section 8.2. The client verifies the response as specified in Section 8.4.

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

```
        |       |       |         Payload: {Code:2.05,
        |       |       |                   Content-Format:0, "180"}
        |       |       |
        |<------+       |            Code: 2.05 (Content)
        |  2.05 |       |           Token: 0x83
        |       |       |         Observe: 8
        |       |       |          OSCORE: [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 dummy Codes (FETCH/Content) are used to allow forwarding of Observe messages. 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

Two examples complying with the requirements on the security context parameters (Section 3.3) are given in this section.

B.1. Master Secret Used Once

An application may derive a security context once and use it for the lifetime of a device. For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device. This specification describes techniques for persistent storage of the security context and synchronization of sequence numbers (see Section 7.5) to ensure that security is maintained with the existing security context.

B.2. Master Secret Used Multiple Times

Section 12.2 recommends the use of a key establishment protocol providing forward secrecy of the Master Secret.

An application which does not require forward secrecy may allow multiple security contexts to be derived from one Master Secret. The requirements on the security context parameters must be fulfilled (Section 3.3) even if the client or server is rebooted, recommissioned or in error cases.

This section gives an example of an application allowing new security contexts to be derived from input parameters pre-established between client and server for this purpose: in particular Master Secret, Master Salt and Sender/Recipient ID (see Section 3.2):

- The client generates an ID Context which has previously not been used with the pre-established input parameters and derives a new security context. ID context may be pseudo-random and large for stochastic uniqueness, but care must be taken e.g. to avoid re-use of the same seed for random number generation. Using this new security context, the client generates an OSCORE request with (kid context, kid) = (ID Context, Sender ID) in the OSCORE option.

- The server receiving such an OSCORE request with kid matching the Recipient ID of pre-established input parameters, but with a new kid context, derives the security context using ID Context = kid context. If the message verifies then a new security context with this ID Context is stored in the server, and used in the response. Further requests with the same (kid context, kid) are verified with this security context.

As an alternative procedure to reduce the subsequent overhead in requests due to kid context, the verification of a message with a new ID Context may trigger the server to generate a new kid to replace the Client Sender ID in future requests. A client may e.g. indicate support for such a procedure by requesting a special well-known URI and receive the new kid in the response, which together with the input parameters and the ID context is used to derive the new security context which may be identified only by its kid. The details are out of scope for this specification.

The procedures may be complemented with the use of the Echo option for verifying the aliveness of the client requesting a new security context.

Appendix C. Test Vectors

This appendix includes the test vectors for different examples of CoAP messages using OSCORE. Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server.

C.1. Test Vector 1: Key Derivation with Master Salt

In this test vector, a Master Salt of 8 bytes is used. The default values are used for AEAD Algorithm and KDF.

C.1.1. Client

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x (0 byte)
- Recipient ID: 0x01 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x8540f60a634b657910 (9 bytes)
- info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)
- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- Recipient Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

C.1.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x01 (1 byte)

o Recipient ID: 0x (0 byte)

From the previous parameters,

o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)

o info (for Recipient Key): 0x8540f60a634b657910 (9 bytes)

o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

o Recipient Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)

o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

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

In this test vector, 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): 0x854100f60a634b657910 (10 bytes)

o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)

o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

o Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)

o Recipient Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)

o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

C.2.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

- Sender ID: 0x01 (1 byte)

- Recipient ID: 0x00 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x854101f60a634b657910 (10 bytes)

- info (for Recipient Key): 0x854100f60a634b657910 (10 bytes)

- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)

- Recipient Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)

- Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

C.3. Test Vector 3: Key Derivation with ID Context

In this test vector, a Master Salt of 8 bytes and a ID Context of 8 bytes are used. The default values are used for AEAD Algorithm and KDF.

C.3.1. Client

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

- Master Salt: 0x9e7ca92223786340 (8 bytes)

- Sender ID: 0x (0 byte)

- Recipient ID: 0x01 (1 byte)

- ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- info (for Sender Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)

- info (for Recipient Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)

- info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)

- Recipient Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)

- Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

C.3.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)

- Master Salt: 0x9e7ca92223786340 (8 bytes)

- Sender ID: 0x01 (1 byte)

- Recipient ID: 0x (0 byte)

- ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- info (for Sender Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)

- info (for Recipient Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)

- info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- Sender Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)

- Recipient Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)

o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

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

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

Unprotected CoAP request:
0x44015d1f00003974396c6f63616c686f737483747631 (22 bytes)

Common Context:

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

o Key Derivation Function: HKDF SHA-256

o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

o Sender ID: 0x (0 byte)

o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (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: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)

o nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

o OSCORE option value: 0x0914 (2 bytes)

o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f6
  3616c686f7374620914ff612f1092f1776f1c1668b3825e (35 bytes)

C.5. Test Vector 5: OSCORE Request, Client

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

Unprotected CoAP request:
0x440171c30000b932396c6f63616c686f737483747631 (22 bytes)

Common Context:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

Sender Context:

- Sender ID: 0x00 (1 bytes)
- Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x00 (1 byte)
- external_aad: 0x8501810a4100411440 (9 bytes)
- AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- nonce: 0xbf35ae297d2dace910c52e99ed (13 bytes)

From the previous parameter, the following is derived:

o OSCORE option value: 0x091400 (3 bytes)

o ciphertext: 0x4ed339a5a379b0b8bc731fffb0 (13 bytes)

From there:

o Protected CoAP request (OSCORE message): 0x440271c30000b932396c6f63616c686f737463091400ff4ed339a5a379b0b8bc731fffb0 (36 bytes)

C.6. Test Vector 6: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request carrying the ID Context in the message, using the security context derived in Appendix C.3. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:
0x44012f8eef9bbf7a396c6f63616c686f737483747631 (22 bytes)

Common Context:

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

o Key Derivation Function: HKDF SHA-256

o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

o ID Context: 0x37cbf3210017a2d3 (8 bytes)

Sender Context:

o Sender ID: 0x (0 bytes)

o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (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 kid context: 0x37cbf3210017a2d3 (8 bytes)

o external_aad: 0x8501810a40411440 (8 bytes)

o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)

o plaintext: 0x01b3747631 (5 bytes)

o encryption key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)

o nonce: 0x2ca58fb85ff1b81c0b7181b84a (13 bytes)

From the previous parameter, the following is derived:

o OSCORE option value: 0x19140837cbf3210017a2d3 (11 bytes)

o ciphertext: 0x72cd7273fd331ac45cffbe55c3 (13 bytes)

From there:

o Protected CoAP request (OSCORE message): 0x44022f8eef9bbf7a396c6f6
3616c686f73746b19140837cbf3210017a2d3ff4ed339a5a379b0b8bc731fffb0
(44 bytes)

C.7. Test Vector 7: OSCORE Response, Server

This section contains a test vector for an OSCORE protected 2.05 Content response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a kid nor a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

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

o Key Derivation Function: HKDF SHA-256

o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

o Sender ID: 0x01 (1 byte)

o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

o external_aad: 0x8501810a40411440 (8 bytes)

- AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)

- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)

- encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x (0 bytes)

- ciphertext: 0xdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (22 bytes)

From there:

- Protected CoAP response (OSCORE message): 0x64445d1f0000397490ffdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (32 bytes)

C.8. Test Vector 8: OSCORE Response with Partial IV, Server

This section contains a test vector for an OSCORE protected 2.05 Content response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a kid, but contains a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

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

- Key Derivation Function: HKDF SHA-256

- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- Sender ID: 0x01 (1 byte)

- Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x00 (1 byte)
- external_aad: 0x8501810a40411440 (8 bytes)
- AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x0100 (2 bytes)
- ciphertext: 0x4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (22 bytes)

From there:

- Protected CoAP response (OSCORE message): 0x64445d1f00003974920100 ff4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (34 bytes)

Appendix D. Overview of Security Properties

D.1. Supporting Proxy Operations

CoAP is designed to work with intermediaries reading and/or changing CoAP message fields to perform supporting operations in constrained environments, e.g. forwarding and cross-protocol translations.

Securing CoAP on transport layer protects the entire message between the endpoints in which case CoAP proxy operations are not possible. In order to enable proxy operations, security on transport layer needs to be terminated at the proxy in which case the CoAP message in its entirety is unprotected in the proxy.

Requirements for CoAP end-to-end security are specified in [I-D.hartke-core-e2e-security-reqs]. The client and server are assumed to be honest, but proxies and gateways are only trusted to perform their intended operations. Forwarding is specified in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. HTTP-CoAP translation is specified in [RFC8075]. Intermediaries translating between different transport layers are intended to perform just that.

By working at the CoAP layer, OSCORE enables different CoAP message fields to be protected differently, which allows message fields required for proxy operations to be available to the proxy while message fields intended for the other endpoint remain protected. In the remainder of this section we analyze how OSCORE protects the protected message fields and the consequences of message fields intended for proxy operation being unprotected.

D.2. Protected Message Fields

Protected message fields are included in the Plaintext (Section 5.3) and the Additional Authenticated Data (Section 5.4) of the COSE_Encrypt0 object and encrypted using an AEAD algorithm.

OSCORE depends on a pre-established random Master Secret (Section 12.3) used to derive encryption keys, and a construction for making (key, nonce) pairs unique (Appendix D.3). Assuming this is true, and the keys are used for no more data than indicated in Section 7.2.1, OSCORE should provide the following guarantees:

o Confidentiality: An attacker should not be able to determine the plaintext contents of a given OSCORE message or determine that different plaintexts are related (Section 5.3).

o Integrity: An attacker should not be able to craft a new OSCORE message with protected message fields different from an existing OSCORE message which will be accepted by the receiver.

o Request-response binding: An attacker should not be able to make a client match a response to the wrong request.

o Non-replayability: An attacker should not be able to cause the receiver to accept a message which it has previously received and accepted.

In the above, the attacker is anyone except the endpoints, e.g. a compromised intermediary. Informally, OSCORE provides these properties by AEAD-protecting the plaintext with a strong key and uniqueness of (key, nonce) pairs. AEAD encryption [RFC5116] provides confidentiality and integrity for the data. Response-request binding is provided by including the kid and Partial IV of the request in the AAD of the response. Non-replayability of requests and notifications is provided by using unique (key, nonce) pairs and a replay protection mechanism (application dependent, see Section 7.4).

OSCORE is susceptible to a variety of traffic analysis attacks based on observing the length and timing of encrypted packets. OSCORE does not provide any specific defenses against this form of attack but the

application may use a padding mechanism to prevent an attacker from directly determine the length of the padding. However, information about padding may still be revealed by side-channel attacks observing differences in timing.

D.3. Uniqueness of (key, nonce)

In this section we show that (key, nonce) pairs are unique as long as the requirements in Sections 3.3 and 7.2.1 are followed.

Fix a Common Context (Section 3.1) and an endpoint, called the encrypting endpoint. An endpoint may alternate between client and server roles, but each endpoint always encrypts with the Sender Key of its Sender Context. Sender Keys are (stochastically) unique since they are derived with HKDF using unique Sender IDs, so messages encrypted by different endpoints use different keys. It remains to prove that the nonces used by the fixed endpoint are unique.

Since the Common IV is fixed, the nonces are determined by a Partial IV (PIV) and the Sender ID of the endpoint generating that Partial IV (ID_PIV). The nonce construction (Section 5.2) with the size of the ID_PIV (S) creates unique nonces for different (ID_PIV, PIV) pairs. There are two cases:

A. For requests, and responses with Partial IV (e.g. Observe notifications):

o ID_PIV = Sender ID of the encrypting endpoint

o PIV = current Partial IV of the encrypting endpoint

Since the encrypting endpoint steps the Partial IV for each use, the nonces used in case A are all unique as long as the number of encrypted messages is kept within the required range (Section 7.2.1).

B. For responses without Partial IV (subset of cases with single response to a request):

o ID_PIV = Sender ID of the endpoint generating the request

o PIV = Partial IV of the request

Since the Sender IDs are unique, ID_PIV is different from the Sender ID of the encrypting endpoint. Therefore, the nonces in case B are different compared to nonces in case A, where the encrypting endpoint generated the Partial IV. Since the Partial IV of the request is verified for replay (Section 7.4) associated to this Recipient

Context, PIV is unique for this ID_PIV, which makes all nonces in case B distinct.

D.4. Unprotected Message Fields

This section lists and discusses issues with unprotected message fields.

D.4.1. CoAP Header Fields

- Version. The CoAP version [RFC7252] is not expected to be sensitive to disclose. Currently there is only one CoAP version defined. A change of this parameter is potentially a denial-of-service attack. Future versions of CoAP need to analyze attacks to OSCORE protected messages due to an adversary changing the CoAP version.

- Token/Token Length. The Token field is a client-local identifier for differentiating between concurrent requests [RFC7252]. An eavesdropper reading the token can match requests to responses which can be used in traffic analysis. In particular this is true for notifications, where multiple responses are matched with one request. CoAP proxies are allowed to change Token and Token Length between UDP hops. However, modifications of Token and Token Length during a UDP hop may become a denial-of-service attack, since it may prevent the client to identify to which request the response belongs or to find the correct information to verify integrity of the response.

- Code. The Outer CoAP Code of an OSCORE message is POST or FETCH for requests with corresponding response codes. The use of FETCH reveals no more than what is revealed by the Outer Observe option. Changing the Outer Code may be a denial-of-service attack by causing errors in the proxy processing.

- Type/Message ID. The Type/Message ID fields [RFC7252] reveal information about the UDP transport binding, e.g. an eavesdropper reading the Type or Message ID gain information about how UDP messages are related to each other. CoAP proxies are allowed to change Type and Message ID. These message fields are not present in CoAP over TCP [RFC8323], and does not impact the request/response message. A change of these fields in a UDP hop is a denial-of-service attack. By sending an ACK, an attacker can make the endpoint believe that the other endpoint received the previous message. By sending a RST, an attacker may be able to cancel an observation, make one endpoint believe the other endpoint is alive, or make one endpoint endpoint believe that the other endpoint is missing some context. By changing a NON to a CON, the

attacker can cause the receiving endpoint to respond to messages for which no response was requested.

- Length. This field contain the length of the message [RFC8323] which may be used for traffic analysis. These message fields are not present in CoAP over UDP, and does not impact the request/response message. A change of Length is a denial-of-service attack similar to changing TCP header fields.

D.4.2. CoAP Options

- Max-Age. The Outer Max-Age is set to zero to avoid unnecessary caching of OSCORE error responses. Changing this value thus may cause unnecessary caching. No additional information is carried with this option.

- Proxy-Uri/Proxy-Scheme. These options are used in forward proxy deployments. With OSCORE, the Proxy-Uri option does not contain the Uri-Path/Uri-Query parts of the URI. The other parts of Proxy-Uri cannot be protected since they are allowed to be changed by a forward proxy. The server can verify what scheme is used in the last hop, but not what was requested by the client or what was used in previous hops.

- Uri-Host/Uri-Port. In forward proxy deployments, the Uri-Host/Uri-Port may be changed by an adversary, and the application needs to handle the consequences of that (see Section 4.1.3.2). The Uri-Host may either be omitted, reveal information equivalent to that of the IP address or more privacy-sensitive information, which is discouraged.

- Observe. The Outer Observe option is intended for an OSCORE-unaware proxy to support forwarding of Observe messages, but is ignored by the endpoints since the Inner Observe determines the processing in the endpoints. Since the Partial IV provides absolute ordering of notifications it is not possible for an intermediary to spoof reordering (see Section 4.1.3.5). The size and distributions of notifications over time may reveal information about the content or nature of the notifications.

- Block1/Block2/Size1/Size2. The Outer Block options enables fragmentation of OSCORE messages in addition to segmentation performed by the Inner Block options. The presence of these options indicates a large message being sent and the message size can be estimated and used for traffic analysis. Manipulating these options is a potential denial-of-service attack, e.g. injection of alleged Block fragments. The specification of a maximum size of message, MAX_UNFRAGMENTED_SIZE

(Section 4.1.3.4.2), above which messages will be dropped, is intended as one measure to mitigate this kind of attack.

- No-Response. 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. Modifying or introducing this option is a potential denial-of-service attack against the proxy operations, but since the option has an Inner value its use can be securely agreed between the endpoints. The presence of this option is not expected to reveal any sensitive information about the message exchange.

- OSCORE. The OSCORE option contains information about the compressed COSE header. Changing this field may cause OSCORE verification to fail.

D.4.3. Error and Signaling Messages

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. For this setting, unprotected error messages are allowed as specified to prevent extensive retransmissions. Those error messages can be spoofed or manipulated, which is a potential denial-of-service attack.

Signaling messages used in CoAP over TCP [RFC8323] are intended to be hop-by-hop; spoofing signaling messages can be used as a denial-of-service attack of a TCP connection.

D.4.4. HTTP Message Fields

In contrast to CoAP, where OSCORE does not protect header fields to enable CoAP-CoAP proxy operations, the use of OSCORE with HTTP is restricted to transporting a protected CoAP message over an HTTP hop. Any unprotected HTTP message fields may reveal information about the transport of the OSCORE message and enable various denial-of-service attacks. It is recommended to additionally use TLS [RFC5246] for HTTP hops, which enables encryption and integrity protection of headers, but still leaves some information for traffic analysis.

Appendix E. CDDL Summary

Data structure definitions in the present specification employ the CDDL language for conciseness and precision. CDDL is defined in [I-D.ietf-cbor-cddl], which at the time of writing this appendix is

in the process of completion. As the document is not yet available for a normative reference, the present appendix defines the small subset of CDDL that is being used in the present specification.

Within the subset being used here, a CDDL rule is of the form "name = type", where "name" is the name given to the "type". A "type" can be one of:

- a reference to another named type, by giving its name. The predefined named types used in the present specification are: "uint", an unsigned integer (as represented in CBOR by major type 0); "int", an unsigned or negative integer (as represented in CBOR by major type 0 or 1); "bstr", a byte string (as represented in CBOR by major type 2); "tstr", a text string (as represented in CBOR by major type 3);

- a choice between two types, by giving both types separated by a "/";

- an array type (as represented in CBOR by major type 4), where the sequence of elements of the array is described by giving a sequence of entries separated by commas ",", and this sequence is enclosed by square brackets "[" and "]". Arrays described by an array description contain elements that correspond one-to-one to the sequence of entries given. Each entry of an array description is of the form "name : type", where "name" is the name given to the entry and "type" is the type of the array element corresponding to this entry.

Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Esko Dijk, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Michael Richardson, Jim Schaad, Peter van der Stok, Dave Thaler, Marco Tiloca, William Vignat, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

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