

Workgroup: ACE Working Group

Internet-Draft:

draft-ietf-ace-oscore-profile-19

Published: 6 May 2021

Intended Status: Standards Track

Expires: 7 November 2021

Authors: F. Palombini   L. Seitz   G. Selander   M. Gunnarsson

Ericsson AB   Combitech   Ericsson AB   RISE

**OSCORE Profile of the Authentication and Authorization for Constrained  
Environments Framework**

**Abstract**

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework. It utilizes Object Security for Constrained RESTful Environments (OSCORE) to provide communication security and proof-of-possession for a key owned by the client and bound to an OAuth 2.0 access token.

**Status of This Memo**

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

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

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

This Internet-Draft will expire on 7 November 2021.

**Copyright Notice**

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

- [1. Introduction](#)
  - [1.1. Terminology](#)
- [2. Protocol Overview](#)
- [3. Client-AS Communication](#)
  - [3.1. C-to-AS: POST to token endpoint](#)
  - [3.2. AS-to-C: Access Token](#)
    - [3.2.1. The OSCORE Input Material](#)
- [4. Client-RS Communication](#)
  - [4.1. C-to-RS: POST to authz-info endpoint](#)
    - [4.1.1. The Nonce 1 Parameter](#)
    - [4.1.2. The ace\\_client\\_recipientid Parameter](#)
  - [4.2. RS-to-C: 2.01 \(Created\)](#)
    - [4.2.1. The Nonce 2 Parameter](#)
    - [4.2.2. The ace\\_server\\_recipientid Parameter](#)
  - [4.3. OSCORE Setup](#)
  - [4.4. Access rights verification](#)
- [5. Secure Communication with AS](#)
- [6. Discarding the Security Context](#)
- [7. Security Considerations](#)
- [8. Privacy Considerations](#)
- [9. IANA Considerations](#)
  - [9.1. ACE Profile Registry](#)
  - [9.2. OAuth Parameters Registry](#)
  - [9.3. OAuth Parameters CBOR Mappings Registry](#)
  - [9.4. OSCORE Security Context Parameters Registry](#)
  - [9.5. CWT Confirmation Methods Registry](#)
  - [9.6. JWT Confirmation Methods Registry](#)
  - [9.7. Expert Review Instructions](#)
- [10. References](#)
  - [10.1. Normative References](#)
  - [10.2. Informative References](#)
- [Appendix A. Profile Requirements](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

## 1. Introduction

This document specifies the "coap\_oscure" profile of the ACE framework [[I-D.ietf-ace-oauth-authz](#)]. In this profile, a client and a resource server use the Constrained Application Protocol (CoAP) [[RFC7252](#)] to communicate. The client uses an access token, bound to a symmetric key (the proof-of-possession key) to authorize its access to the resource server. Note that this profile uses a symmetric-crypto-based scheme, where the symmetric secret is used as

input material for keying material derivation. In order to provide communication security and proof of possession, the client and resource server use Object Security for Constrained RESTful Environments (OSCORE) [[RFC8613](#)]. Note that the proof of possession is not achieved through a dedicated protocol element, but rather occurs after the first message exchange using OSCORE.

OSCORE specifies how to use CBOR Object Signing and Encryption (COSE) [[I-D.ietf-cose-rfc8152bis-struct](#)][[I-D.ietf-cose-rfc8152bis-algs](#)] to secure CoAP messages. Note that OSCORE can be used to secure CoAP messages, as well as HTTP and combinations of HTTP and CoAP; a profile of ACE similar to the one described in this document, with the difference of using HTTP instead of CoAP as communication protocol, could be specified analogously to this one.

### 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.

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "Message Authentication Code (MAC)", "Hash-based Message Authentication Code (HMAC)", and "verify" are taken from [[RFC4949](#)].

RESTful terminology follows HTTP [[RFC7231](#)].

Readers are expected to be familiar with the terms and concepts defined in OSCORE [[RFC8613](#)], such as "Security Context" and "Recipient ID".

Terminology for entities in the architecture is defined in OAuth 2.0 [[RFC6749](#)], such as client (C), resource server (RS), and authorization server (AS). It is assumed in this document that a given resource on a specific RS is associated to a unique AS.

Concise Binary Object Representation (CBOR) [[RFC8949](#)] and Concise Data Definition Language (CDDL) [[RFC8610](#)] are used in this document. CDDL predefined type names, especially bstr for CBOR byte strings and tstr for CBOR text strings, are used extensively in this document.

Note that the term "endpoint" is used here, as in [[I-D.ietf-ace-oauth-authz](#)], following its OAuth definition, which is to denote resources such as token and introspect at the AS and authz-info at the RS. The CoAP [[RFC7252](#)] definition, which is "An entity participating in the CoAP protocol" is not used in this document.

Examples throughout this document are expressed in CBOR diagnostic notation without the tag and value abbreviations.

## 2. Protocol Overview

This section gives an overview of how to use the ACE Framework [[I-D.ietf-ace-oauth-authz](#)] to secure the communication between a client and a resource server using OSCORE [[RFC8613](#)]. The parameters needed by the client to negotiate the use of this profile with the authorization server, as well as the OSCORE setup process, are described in detail in the following sections.

The RS maintains a collection of OSCORE Security Contexts with associated authorization information for all the clients that it is communicating with. The authorization information is maintained as policy that is used as input to processing requests from those clients.

This profile requires a client to retrieve an access token from the AS for the resource it wants to access on an RS, by sending an access token request to the token endpoint, as specified in section 5.8 of [[I-D.ietf-ace-oauth-authz](#)]. The access token request and response MUST be confidentiality-protected and ensure authenticity. This profile RECOMMENDS the use of OSCORE between client and AS, to reduce the number of libraries the client has to support, but other protocols fulfilling the security requirements defined in section 5 of [[I-D.ietf-ace-oauth-authz](#)] MAY alternatively be used, such as TLS [[RFC8446](#)] or DTLS [[I-D.ietf-tls-dtls13](#)].

Once the client has retrieved the access token, it generates a nonce N1, defined in this document (see [Section 4.1.1](#)). The client also generates its own OSCORE Recipient ID ID1 (see Section 3.1 of [[RFC8613](#)]), for use with the keying material associated to the RS. The client posts the token, N1 and its Recipient ID to the RS using the authz-info endpoint and mechanisms specified in section 5.8 of [[I-D.ietf-ace-oauth-authz](#)] and Content-Format = application/ace+cbor. When using this profile, the communication with the authz-info endpoint is not protected, except for update of access rights.

If the access token is valid, the RS replies to this request with a 2.01 (Created) response with Content-Format = application/ace+cbor, which contains a nonce N2 and its newly generated OSCORE Recipient ID, ID2, for use with the keying material associated to the client. Moreover, the server concatenates the input salt received in the token, N1, and N2 to obtain the Master Salt of the OSCORE Security Context (see section 3 of [[RFC8613](#)]). The RS then derives the complete Security Context associated with the received token from the Master Salt, the OSCORE Recipient ID generated by the client (set as its OSCORE Sender ID), its own OSCORE Recipient ID, plus the

parameters received in the access token from the AS, following section 3.2 of [[RFC8613](#)].

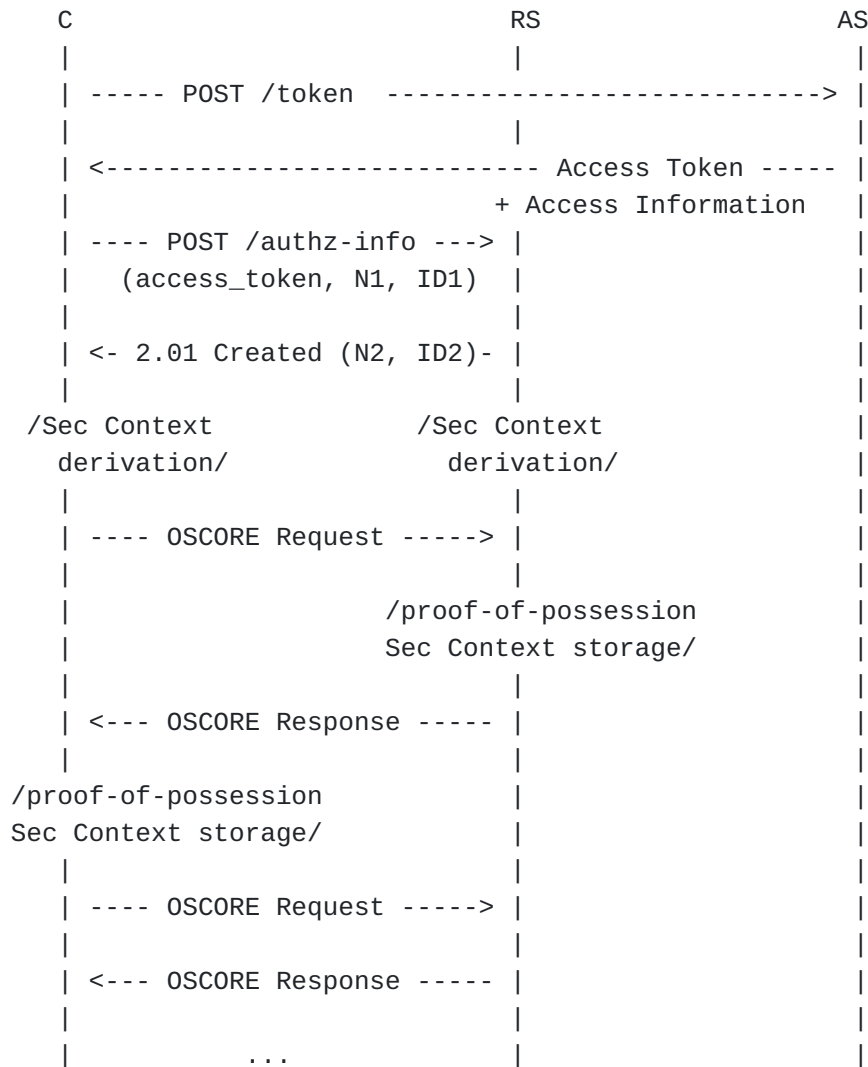
In a similar way, after receiving the nonce N2, the client concatenates the input salt, N1 and N2 to obtain the Master Salt of the OSCORE Security Context. The client then derives the complete Security Context from the Master Salt, the OSCORE Recipient ID generated by the RS (set as its OSCORE Sender ID), its own OSCORE Recipient ID, plus the parameters received from the AS.

Finally, the client starts the communication with the RS by sending a request protected with OSCORE to the RS. If the request is successfully verified, the server stores the complete Security Context state that is ready for use in protecting messages, and uses it in the response, and in further communications with the client, until token deletion due to, for example, expiration. This Security Context is discarded when a token (whether the same or a different one) is used to successfully derive a new Security Context for that client.

The use of nonces N1 and N2 during the exchange prevents the reuse of an Authenticated Encryption with Associated Data (AEAD) nonce/key pair for two different messages. Reuse might otherwise occur when client and RS derive a new Security Context from an existing (non-expired) access token, as might occur when either party has just rebooted, and might lead to loss of both confidentiality and integrity. Instead, by using the exchanged nonces N1 and N2 as part of the Master Salt, the request to the authz-info endpoint posting the same token results in a different Security Context, by OSCORE construction, since even though the Master Secret, Sender ID and Recipient ID are the same, the Master Salt is different (see Section 3.2.1 of [[RFC8613](#)]). If the exchanged nonces were reused, a node reusing a non-expired old token would be susceptible to on-path attackers provoking the creation of an OSCORE message using an old AEAD key and nonce.

After the whole message exchange has taken place, the client can contact the AS to request an update of its access rights, sending a similar request to the token endpoint that also includes an identifier so that the AS can find the correct OSCORE security input material it has previously shared with the client. This specific identifier, encoded as a byte string, is assigned by the AS to be unique in the sets of its OSCORE security input materials, and is not used as input material to derive the full OSCORE Security Context.

An overview of the profile flow for the OSCORE profile is given in [Figure 1](#). The names of messages coincide with those of [[I-D.ietf-ace-oauth-authz](#)] when applicable.



### Figure 1: Protocol Overview

### 3. Client-AS Communication

The following subsections describe the details of the POST request and response to the token endpoint between client and AS. Section 3.2 of [RFC8613] defines how to derive a Security Context based on a shared master secret and a set of other parameters, established between client and server, which the client receives from the AS in this exchange. The proof-of-possession key (pop-key) included in the response from the AS MUST be used as master secret in OSCORE.

### 3.1. C-to-AS: POST to token endpoint

The client-to-AS request is specified in Section 5.8.1 of [[I-D.ietf-ace-oauth-authz](#)].

The client must send this POST request to the token endpoint over a secure channel that guarantees authentication, message integrity and confidentiality (see [Section 5](#)).

An example of such a request is shown in [Figure 2](#)

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "audience" : "tempSensor4711",
  "scope" : "read"
}
```

Figure 2: Example C-to-AS POST /token request for an access token bound to a symmetric key.

If the client wants to update its access rights without changing an existing OSCORE Security Context, it MUST include in its POST request to the token endpoint a req\_cnf object, with the kid field carrying a CBOR byte string containing the OSCORE Input Material Identifier (assigned as discussed in [Section 3.2](#)). This identifier, together with other information such as audience (see Section 5.8.1 of [\[I-D.ietf-ace-oauth-authz\]](#)), can be used by the AS to determine the shared secret bound to the proof-of-possession token and therefore MUST identify a symmetric key that was previously generated by the AS as a shared secret for the communication between the client and the RS. The AS MUST verify that the received value identifies a proof-of-possession key that has previously been issued to the requesting client. If that is not the case, the Client-to-AS request MUST be declined with the error code `invalid_request` as defined in Section 5.8.3 of [\[I-D.ietf-ace-oauth-authz\]](#).

An example of such a request is shown in [Figure 3](#)

```

Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "audience" : "tempSensor4711",
  "scope" : "write",
  "req_cnf" : {
    "kid" : h'01'
  }
}

```

Figure 3: Example C-to-AS POST /token request for updating rights to an access token bound to a symmetric key.

### 3.2. AS-to-C: Access Token

After verifying the POST request to the token endpoint and that the client is authorized to obtain an access token corresponding to its access token request, the AS responds as defined in section 5.8.2 of [\[I-D.ietf-ace-oauth-authz\]](#). If the client request was invalid, or not authorized, the AS returns an error response as described in section 5.8.3 of [\[I-D.ietf-ace-oauth-authz\]](#).

The AS can signal that the use of OSCORE is REQUIRED for a specific access token by including the "ace\_profile" parameter with the value "coap\_oscore" in the access token response. This means that the client MUST use OSCORE towards all resource servers for which this access token is valid, and follow [Section 4.3](#) to derive the security context to run OSCORE. Usually it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile signaling can be omitted.

Moreover, the AS MUST send the following data:

- \*a master secret
- \*an identifier of the OSCORE Input Material

Additionally, the AS MAY send the following data, in the same response.

- \*a context identifier
- \*an AEAD algorithm
- \*an HMAC-based key derivation function (HKDF, [\[RFC5869\]](#)) algorithm, see section 3.1 of [\[I-D.ietf-cose-rfc8152bis-algs\]](#)



\*a salt

\*the OSCORE version number

This data is transported in the OSCORE\_Input\_Material. The OSCORE\_Input\_Material is a CBOR map object, defined in [Section 3.2.1](#). This object is transported in the cnf parameter of the access token response as defined in Section 3.2 of [[I-D.ietf-ace-oauth-params](#)], as the value of a field named osc, registered in [Section 9.5](#) and [Section 9.6](#).

The AS MAY assign an identifier to the context (context identifier). This identifier is used as ID Context in the OSCORE context as described in section 3.1 of [[RFC8613](#)]. If assigned, this parameters MUST be communicated as the contextId field in the OSCORE\_Input\_Material. The application needs to consider that this identifier is sent in the clear and may reveal information about the endpoints, as mentioned in section 12.8 of [[RFC8613](#)].

The master secret and the identifier of the OSCORE\_Input\_Material MUST be communicated as the ms and id field in the osc field in the cnf parameter of the access token response. If included, the AEAD algorithm is sent in the alg parameter in the OSCORE\_Input\_Material; the HKDF algorithm in the hkdf parameter of the OSCORE\_Input\_Material; a salt in the salt parameter of the OSCORE\_Input\_Material; and the OSCORE version in the version parameter of the OSCORE\_Input\_Material.

The same parameters MUST be included in the claims associated with the access token. The OSCORE master secret MUST be encrypted by the authorization server so that only the resource server can decrypt it (see Section 6.1. of [[I-D.ietf-ace-oauth-authz](#)]). This profile RECOMMENDS the use of a CBOR web token (CWT) protected with COSE\_Encrypt/COSE\_Encrypt0 as specified in [[RFC8392](#)]. If the token is a CWT, the same OSCORE\_Input\_Material structure defined above MUST be placed in the osc field of the cnf claim of this token.

The AS MUST send different OSCORE\_Input\_Material (and therefore different access tokens) to different authorized clients, in order for the RS to differentiate between clients.

[Figure 4](#) shows an example of an AS response. The access token has been truncated for readability.

```

Header: Created (Code=2.01)
Content-Type: "application/ace+cbor"
Payload:
{
  "access_token" : h'8343a1010aa2044c53 ...
    (remainder of access token (CWT) omitted for brevity)',
  "ace_profile" : "coap_oscure",
  "expires_in" : "3600",
  "cnf" : {
    "osc" : {
      "id" : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f'
    }
  }
}

```

Figure 4: Example AS-to-C Access Token response with OSCORE profile.

[Figure 5](#) shows an example CWT Claims Set, including the relevant OSCORE parameters in the cnf claim.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "osc" : {
      "ms" : h'f9af838368e353e78888e1426bd94e6f',
      "id" : h'01'
    }
  }
}

```

Figure 5: Example CWT Claims Set with OSCORE parameters.

The same CWT Claims Set as in [Figure 5](#), using the value abbreviations defined in [[I-D.ietf-ace-oauth-authz](#)] and [[RFC8747](#)] and encoded in CBOR is shown in [Figure 6](#). The bytes in hexadecimal are reported in the first column, while their corresponding CBOR meaning is reported after the # sign on the second column, for easiness of readability.

NOTE TO THE RFC EDITOR: before publishing, it should be checked (and in case fixed) that the values used below (which are not yet registered) are the final values registered in IANA.

```

A5                                     # map(5)
63                                     # text(3)
  617564                             # "aud"
76                                     # text(22)
  74656D7053656E736F72496E4C6976696E67526F6F6D
                                     # "tempSensorInLivingRoom"
63                                     # text(3)
  696174                             # "iat"
6A                                     # text(10)
  31333630313839323234              # "1360189224"
63                                     # text(3)
  657870                             # "exp"
6A                                     # text(10)
  31333630323839323234              # "1360289224"
65                                     # text(5)
  73636F7065                       # "scope"
78 18                               # text(24)
  74656D70657261747572655F67206669726D7761726555F70
                                     # "temperature_g firmware_p"
63                                     # text(3)
  636E66                             # "cnf"
A1                                     # map(1)
  63                                     # text(3)
    6F7363                          # "osc"
A2                                     # map(2)
  62                                     # text(2)
    6D73                             # "ms"
  50                                     # bytes(16)
    F9AF838368E353E78888E1426BD94E6F
                                     # "\xF9\xAF\x83\x83h\xE3S\xE7
                                     \x88\x88\xE1Bk\xD9No"
  62                                     # text(2)
    6964                             # "id"
  41                                     # bytes(1)
    01                               # "\x01"

```

Figure 6: Example CWT Claims Set with OSCORE parameters, CBOR encoded.

If the client has requested an update to its access rights using the same OSCORE Security Context, which is valid and authorized, the AS MUST omit the cnf parameter in the response, and MUST carry the OSCORE Input Material identifier in the kid field in the cnf claim of the token. This identifier needs to be included in the token in order for the RS to identify the correct OSCORE Input Material.

[Figure 7](#) shows an example of such an AS response. The access token has been truncated for readability.

```

Header: Created (Code=2.01)
Content-Type: "application/ace+cbor"
Payload:
{
  "access_token" : h'8343a1010aa2044c53 ...
    (remainder of access token (CWT) omitted for brevity)',
  "ace_profile" : "coap_oscure",
  "expires_in" : "3600"
}

```

Figure 7: Example AS-to-C Access Token response with OSCORE profile, for update of access rights.

[Figure 8](#) shows an example CWT Claims Set, containing the necessary OSCORE parameters in the cnf claim for update of access rights.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_h",
  "cnf" : {
    "kid" : h'01'
  }
}

```

Figure 8: Example CWT Claims Set with OSCORE parameters for update of access rights.

### 3.2.1. The OSCORE\_Input\_Material

An OSCORE\_Input\_Material is an object that represents the input material to derive an OSCORE Security Context, i.e., the local set of information elements necessary to carry out the cryptographic operations in OSCORE (Section 3.1 of [\[RFC8613\]](#)). In particular, the OSCORE\_Input\_Material is defined to be serialized and transported between nodes, as specified by this document, but can also be used by other specifications if needed. The OSCORE\_Input\_Material can either be encoded as a JSON object or as a CBOR map. The set of common parameters that can appear in an OSCORE\_Input\_Material can be found in the IANA "OSCORE Security Context Parameters" registry ([Section 9.4](#)), defined for extensibility, and the initial set of parameters defined in this document is specified below. All parameters are optional. [Table 1](#) provides a summary of the OSCORE\_Input\_Material parameters defined in this section.

name	CBOR label	CBOR type	registry	description
id	0	byte string		OSCORE Input Material Identifier
version	1	unsigned integer		OSCORE Version
ms	2	byte string		OSCORE Master Secret value
hkdf	3	text string / integer	<a href="#">[COSE.Algorithms]</a> Values (HMAC-based)	OSCORE HKDF value
alg	4	text string / integer	<a href="#">[COSE.Algorithms]</a> Values (AEAD)	OSCORE AEAD Algorithm value
salt	5	byte string		an input to OSCORE Master Salt value
contextId	6	byte string		OSCORE ID Context value

Table 1: OSCORE\_Input\_Material Parameters

**id:** This parameter identifies the OSCORE\_Input\_Material and is encoded as a byte string. In JSON, the "id" value is a Base64 encoded byte string. In CBOR, the "id" type is byte string, and has label 0.

**version:** This parameter identifies the OSCORE Version number, which is an unsigned integer. For more information about this field, see section 5.4 of [\[RFC8613\]](#). In JSON, the "version" value is an integer. In CBOR, the "version" type is integer, and has label 1.

**ms:** This parameter identifies the OSCORE Master Secret value, which is a byte string. For more information about this field, see section 3.1 of [\[RFC8613\]](#). In JSON, the "ms" value is a Base64 encoded byte string. In CBOR, the "ms" type is byte string, and has label 2.

**hkdf:** This parameter identifies the OSCORE HKDF Algorithm. For more information about this field, see section 3.1 of [\[RFC8613\]](#). The values used MUST be registered in the IANA "COSE Algorithms" registry (see [\[COSE.Algorithms\]](#)) and MUST be HMAC-based HKDF algorithms (see section 3.1 of [\[I-D.ietf-cose-rfc8152bis-algs\]](#)). The value can either be the integer or the text string value of the HMAC-based HKDF algorithm in the "COSE Algorithms" registry. In JSON, the "hkdf" value is a case-sensitive ASCII string or an integer. In CBOR, the "hkdf" type is text string or integer, and has label 3.

**alg:**

This parameter identifies the OSCORE AEAD Algorithm. For more information about this field, see section 3.1 of [[RFC8613](#)] The values used MUST be registered in the IANA "COSE Algorithms" registry (see [[COSE.Algorithms](#)]) and MUST be AEAD algorithms. The value can either be the integer or the text string value of the HMAC-based HKDF algorithm in the "COSE Algorithms" registry. In JSON, the "alg" value is a case-sensitive ASCII string or an integer. In CBOR, the "alg" type is text string or integer, and has label 4.

**salt:** This parameter identifies an input to the OSCORE Master Salt value, which is a byte string. For more information about this field, see section 3.1 of [[RFC8613](#)]. In JSON, the "salt" value is a Base64 encoded byte string. In CBOR, the "salt" type is byte string, and has label 5.

**contextId:** This parameter identifies the security context as a byte string. This identifier is used as OSCORE ID Context. For more information about this field, see section 3.1 of [[RFC8613](#)]. In JSON, the "contextID" value is a Base64 encoded byte string. In CBOR, the "contextID" type is byte string, and has label 6.

An example of JSON OSCORE\_Input\_Material is given in [Figure 9](#).

```
"osc" : {
  "alg" : "AES-CCM-16-64-128",
  "id" : b64'AQ=='
  "ms" : b64'+aDg2jjU+eIi0FCa9l0bw'
}
```

Figure 9: Example JSON OSCORE\_Input\_Material

The CDDL grammar describing the CBOR OSCORE\_Input\_Material is:

```
OSCORE_Input_Material = {
  ? 0 => bstr,           ; id
  ? 1 => int,             ; version
  ? 2 => bstr,            ; ms
  ? 3 => tstr / int,      ; hkdf
  ? 4 => tstr / int,      ; alg
  ? 5 => bstr,            ; salt
  ? 6 => bstr,            ; contextId
  * int / tstr => any
}
```

## 4. Client-RS Communication

The following subsections describe the details of the POST request and response to the authz-info endpoint between client and RS. The client generates a nonce N1 and an identifier ID1 unique in the sets of its own Recipient IDs, and posts them together with the token that includes the materials (e.g., OSCORE parameters) received from the AS to the RS. The RS then generates a nonce N2 and an identifier ID2 unique in the sets of its own Recipient IDs, and uses Section 3.2 of [[RFC8613](#)] to derive a security context based on a shared master secret, the two exchanged nonces and the two identifiers, established between client and server. The exchanged nonces and identifiers are encoded as CBOR byte string if CBOR is used, and as Base64 string if JSON is used. This security context is used to protect all future communication between client and RS using OSCORE, as long as the access token is valid.

Note that the RS and client authenticate each other by generating the shared OSCORE Security Context using the pop-key as master secret. An attacker posting a valid token to the RS will not be able to generate a valid OSCORE Security Context and thus not be able to prove possession of the pop-key. Additionally, the mutual authentication is only achieved after the client has successfully verified a response from the RS protected with the generated OSCORE Security Context.

### 4.1. C-to-RS: POST to authz-info endpoint

The client MUST generate a nonce value N1 very unlikely to have been previously used with the same input keying material. This profile RECOMMENDS using a 64-bit long random number as the nonce's value. The client MUST store the nonce N1 as long as the response from the RS is not received and the access token related to it is still valid (to the best of the client's knowledge).

The client generates its own Recipient ID, ID1, for the OSCORE Security Context that it is establishing with the RS. By generating its own Recipient ID, the client makes sure that it does not collide with any of its Recipient IDs, nor with any other identifier ID1 if the client is executing this exchange with a different RS at the same time.

The client MUST use CoAP and the Authorization Information resource as described in section 5.8.1 of [[I-D.ietf-ace-oauth-authz](#)] to transport the token, N1 and ID1 to the RS.

Note that the use of the payload and the Content-Format is different from what is described in section 5.8.1 of [[I-D.ietf-ace-oauth-authz](#)], which only transports the token without any CBOR wrapping.

In this profile, the client MUST wrap the token, N1 and ID1 in a CBOR map. The client MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [[I-D.ietf-ace-oauth-authz](#)]. The client MUST include the access token using the `access_token` parameter, N1 using the `nonce1` parameter defined in [Section 4.1.1](#), and ID1 using the `ace_client_recipientid` parameter defined in [Section 4.1.2](#).

The communication with the `authz-info` endpoint does not have to be protected, except for the update of access rights case described below.

Note that a client may be required to re-POST the access token in order to complete a request, since an RS may delete a stored access token (and associated Security Context) at any time, for example due to all storage space being consumed. This situation is detected by the client when it receives an AS Request Creation Hints response. Reposting the same access token will result in deriving a new OSCORE Security Context to be used with the RS, as different exchanged nonces will be used.

The client may also choose to re-POST the access token in order to update its OSCORE Security Context. In that case, the client and the RS will exchange newly generated nonces, re-negotiate identifiers, and derive new keying material. The client and RS might decide to keep the same identifiers or renew them during the re-negotiation.

[Figure 10](#) shows an example of the request sent from the client to the RS. The access token has been truncated for readability.

```
Header: POST (Code=0.02)
Uri-Host: "rs.example.com"
Uri-Path: "authz-info"
Content-Format: "application/ace+cbor"
Payload:
{
  "access_token": h'8343a1010aa2044c53 ...
  (remainder of access token (CWT) omitted for brevity)',
  "nonce1": h'018a278f7faab55a',
  "ace_client_recipientid" : h'1645'
}
```

Figure 10: Example C-to-RS POST `/authz-info` request using CWT

If the client has already posted a valid token, has already established a security association with the RS, and wants to update its access rights, the client can do so by posting the new token (retrieved from the AS and containing the update of access rights)



to the /authz-info endpoint. The client MUST protect the request using the OSCORE Security Context established during the first token exchange. The client MUST only send the access\_token field in the CBOR map in the payload, no nonce or identifier are sent. After proper verification (see [Section 4.2](#)), the RS will replace the old token with the new one, maintaining the same Security Context.

#### **4.1.1. The Nonce 1 Parameter**

This parameter MUST be sent from the client to the RS, together with the access token, if the ace profile used is coap\_oscore, and the message is not an update of access rights, protected with an existing OSCORE Security Context. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (Base64 encoded binary) for JSON-based interactions. This parameter is registered in [Section 9.2](#).

#### **4.1.2. The ace\_client\_recipientid Parameter**

This parameter MUST be sent from the client to the RS, together with the access token, if the ace profile used is coap\_oscore, and the message is not an update of access rights, protected with an existing OSCORE Security Context. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (Base64 encoded binary) for JSON-based interactions. This parameter is registered in [Section 9.2](#).

### **4.2. RS-to-C: 2.01 (Created)**

The RS MUST follow the procedures defined in section 5.8.1 of [[I-D.ietf-ace-oauth-authz](#)]: the RS must verify the validity of the token. If the token is valid, the RS must respond to the POST request with 2.01 (Created). If the token is valid but is associated to claims that the RS cannot process (e.g., an unknown scope), or if any of the expected parameters is missing (e.g., any of the mandatory parameters from the AS or the identifier id1), or if any parameters received in the osc field is unrecognized, the RS must respond with an error response code equivalent to the CoAP code 4.00 (Bad Request). In the latter two cases, the RS may provide additional information in the error response, in order to clarify what went wrong. The RS may make an introspection request (see Section 5.9.1 of [[I-D.ietf-ace-oauth-authz](#)]) to validate the token before responding to the POST request to the authz-info endpoint.

Additionally, the RS MUST generate a nonce N2 very unlikely to have been previously used with the same input keying material, and its own Recipient ID, ID2. The RS makes sure that ID2 does not collide with any of its Recipient IDs. The RS MUST ensure that ID2 is different from the value received in the ace\_client\_recipientid

parameter. The RS sends N2 and ID2 within the 2.01 (Created) response. The payload of the 2.01 (Created) response MUST be a CBOR map containing the nonce2 parameter defined in [Section 4.2.1](#), set to N2, and the ace\_server\_recipientid parameter defined in [Section 4.2.2](#), set to ID2. This profile RECOMMENDS using a 64-bit long random number as the nonce's value. The RS MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [\[I-D.ietf-ace-oauth-authz\]](#).

[Figure 11](#) shows an example of the response sent from the RS to the client.

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "nonce2": h'25a8991cd700ac01',
  "ace_server_recipientid" : h'0000'
}
```

Figure 11: Example RS-to-C 2.01 (Created) response

As specified in section 5.8.3 of [\[I-D.ietf-ace-oauth-authz\]](#), the RS must notify the client with an error response with code 4.01 (Unauthorized) for any long running request before terminating the session, when the access token expires.

If the RS receives the token in a OSCORE protected message, it means that the client is requesting an update of access rights. The RS MUST ignore any nonce and identifiers in the request, if any was sent. The RS MUST check that the "kid" of the cnf claim of the new access token matches the identifier of the OSCORE Input Material of the context used to protect the message. If that is the case, the RS MUST overwrite the old token and associate the new token to the Security Context identified by the "kid" value in the cnf claim. The RS MUST respond with a 2.01 (Created) response protected with the same Security Context, with no payload. If any verification fails, the RS MUST respond with a 4.01 (Unauthorized) error response.

As specified in section 5.8.1 of [\[I-D.ietf-ace-oauth-authz\]](#), when receiving an updated access token with updated authorization information from the client (see [Section 3.1](#)), it is recommended that the RS overwrites the previous token, that is only the latest authorization information in the token received by the RS is valid. This simplifies the process needed by the RS to keep track of authorization information for a given client.

#### 4.2.1. The Nonce 2 Parameter

This parameter MUST be sent from the RS to the client if the ace profile used is `coap_oscure`, and the message is not a response to an update of access rights, protected with an existing OSCORE Security Context. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (Base64 encoded binary) for JSON-based interactions. This parameter is registered in [Section 9.2](#)

#### 4.2.2. The `ace_server_recipientid` Parameter

This parameter MUST be sent from the RS to the client if the ace profile used is `coap_oscure`, and the message is not a response to an update of access rights, protected with an existing OSCORE Security Context. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (Base64 encoded binary) for JSON-based interactions. This parameter is registered in [Section 9.2](#)

#### 4.3. OSCORE Setup

Once the 2.01 (Created) response is received from the RS, following the POST request to `authz-info` endpoint, the client MUST extract the `bstr` nonce N2 from the `nonce2` parameter in the CBOR map in the payload of the response. Then, the client MUST set the Master Salt of the Security Context created to communicate with the RS to the concatenation of salt, N1, and N2, in this order: Master Salt = salt | N1 | N2, where | denotes byte string concatenation, where salt is the CBOR byte string received from the AS in [Section 3.2](#), and where N1 and N2 are the two nonces encoded as CBOR byte strings. An example of Master Salt construction using CBOR encoding is given in [Figure 12](#).

N1, N2 and input salt expressed in CBOR diagnostic notation:

```
nonce1 = h'018a278f7faab55a'  
nonce2 = h'25a8991cd700ac01'  
input salt = h'f9af838368e353e78888e1426bd94e6f'
```

N1, N2 and input salt as CBOR encoded byte strings:

```
nonce1 = 0x48018a278f7faab55a  
nonce2 = 0x4825a8991cd700ac01  
input salt = 0x50f9af838368e353e78888e1426bd94e6f
```

```
Master Salt = 0x50 f9af838368e353e78888e1426bd94e6f  
48 018a278f7faab55a 48 25a8991cd700ac01
```

Figure 12: Example of Master Salt construction using CBOR encoding

If JSON is used instead of CBOR, the Master Salt of the Security Context is the Base64 encoding of the concatenation of the same parameters, each of them prefixed by their size, encoded in 1 byte. When using JSON, the nonces and input salt have a maximum size of 255 bytes. An example of Master Salt construction using Base64 encoding is given in [Figure 13](#).

N1, N2 and input salt values:

```
nonce1 = 0x018a278f7faab55a (8 bytes)
nonce2 = 0x25a8991cd700ac01 (8 bytes)
input salt = 0xf9af838368e353e78888e1426bd94e6f (16 bytes)
```

Input to Base64 encoding: 0x10 f9af838368e353e78888e1426bd94e6f  
08 018a278f7faab55a 08 25a8991cd700ac01

Master Salt = b64'EPmvg4No41PniIjhQmvZTm8IAYonj3+qtVoIJaiZHNcArAE='

Figure 13: Example of Master Salt construction using Base64 encoding

The client MUST set the Sender ID to the `ace_server_recipientid` received in [Section 4.2](#), and the Recipient ID to the `ace_client_recipientid` sent in [Section 4.1](#). The client MUST set the Master Secret from the parameter received from the AS in [Section 3.2](#). The client MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version from the parameters received from the AS in [Section 3.2](#), if present. In case an optional parameter is omitted, the default value SHALL be used as described in sections 3.2 and 5.4 of [\[RFC8613\]](#). After that, the client MUST derive the complete Security Context following section 3.2.1 of [\[RFC8613\]](#). From this point on, the client MUST use this Security Context to communicate with the RS when accessing the resources as specified by the authorization information.

If any of the expected parameters is missing (e.g., any of the mandatory parameters from the AS or the RS), or if `ace_client_recipientid` equals `ace_server_recipientid` (and as a consequence the Sender and Recipient Keys derived would be equal, see section 3.3 of [\[RFC8613\]](#)), then the client MUST stop the exchange, and MUST NOT derive the Security Context. The client MAY restart the exchange, to get the correct security material.

The client then uses this Security Context to send requests to the RS using OSCORE.

After sending the 2.01 (Created) response, the RS MUST set the Master Salt of the Security Context created to communicate with the client to the concatenation of salt, N1, and N2, in the same way described above. An example of Master Salt construction using CBOR

encoding is given in [Figure 12](#) and using Base64 encoding is given in [Figure 13](#). The RS MUST set the Sender ID from the `ace_client_recipientid` received in [Section 4.1](#), and the Recipient ID from the `ace_server_recipientid` sent in [Section 4.2](#). The RS MUST set the Master Secret from the parameter received from the AS and forwarded by the client in the access token in [Section 4.1](#) after validation of the token as specified in [Section 4.2](#). The RS MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version from the parameters received from the AS and forwarded by the client in the access token in [Section 4.1](#) after validation of the token as specified in [Section 4.2](#), if present. In case an optional parameter is omitted, the default value SHALL be used as described in sections 3.2 and 5.4 of [\[RFC8613\]](#). After that, the RS MUST derive the complete Security Context following section 3.2.1 of [\[RFC8613\]](#), and MUST associate this Security Context with the authorization information from the access token.

The RS then uses this Security Context to verify requests and send responses to the client using OSCORE. If OSCORE verification fails, error responses are used, as specified in section 8 of [\[RFC8613\]](#). Additionally, if OSCORE verification succeeds, the verification of access rights is performed as described in section [Section 4.4](#). The RS MUST NOT use the Security Context after the related token has expired, and MUST respond with a unprotected 4.01 (Unauthorized) error message to requests received that correspond to a Security Context with an expired token.

Note that the ID Context can be assigned by the AS, communicated and set in both the RS and client after the exchange specified in this profile is executed. Subsequently, client and RS can update their ID Context by running a mechanism such as the one defined in Appendix B.2 of [\[RFC8613\]](#) if they both support it and are configured to do so. In that case, the ID Context in the OSCORE Security Context will not match the "contextId" parameter of the corresponding OSCORE\_Input\_Material. Running Appendix B.2 results in the keying material in the Security Contexts of client and RS being updated; this same result can also be achieved by the client reposting the access token to the unprotected /authz-info endpoint at the RS, as described in [Section 4.1](#), but without updating the ID Context.

#### **4.4. Access rights verification**

The RS MUST follow the procedures defined in section 5.8.2 of [\[I-D.ietf-ace-oauth-authz\]](#): if an RS receives an OSCORE-protected request from a client, then the RS processes it according to [\[RFC8613\]](#). If OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the authorization information using the access token associated to the Security

Context. The RS then must verify that the authorization information covers the resource and the action requested.

## 5. Secure Communication with AS

As specified in the ACE framework (section 5.9 of [[I-D.ietf-ace-oauth-authz](#)]), the requesting entity (RS and/or client) and the AS communicates via the introspection or token endpoint. The use of CoAP and OSCORE ([[RFC8613](#)]) for this communication is RECOMMENDED in this profile; other protocols fulfilling the security requirements defined in section 5 of [[I-D.ietf-ace-oauth-authz](#)] (such as HTTP and DTLS or TLS) MAY be used instead.

If OSCORE is used, the requesting entity and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the requesting entity and the AS communicate through the introspection endpoint as specified in section 5.9 of [[I-D.ietf-ace-oauth-authz](#)] and through the token endpoint as specified in section 5.8 of [[I-D.ietf-ace-oauth-authz](#)].

## 6. Discarding the Security Context

There are a number of scenarios where a client or RS needs to discard the OSCORE security context, and acquire a new one.

The client MUST discard the current Security Context associated with an RS when any of the following occurs:

- \*the Sequence Number space ends.
- \*the access token associated with the context becomes invalid due to, for example, expiration.
- \*the client receives a number of 4.01 Unauthorized responses to OSCORE requests using the same Security Context. The exact number needs to be specified by the application.
- \*the client receives a new nonce in the 2.01 (Created) response (see [Section 4.2](#)) to a POST request to the authz-info endpoint, when re-posting a (non-expired) token associated to the existing context.

The RS MUST discard the current Security Context associated with a client when any of the following occurs:

- \*the Sequence Number space ends.
- \*the access token associated with the context expires.

\*the client has successfully replaced the current security context with a newer one by posting an access token to the unprotected / authz-info endpoint at the RS, e.g., by re-posting the same token, as specified in [Section 4.1](#).

Whenever one more access token is successfully posted to the RS, and a new Security Context is derived between the client and RS, messages in transit that were protected with the previous Security Context might not pass verification, as the old context is discarded. That means that messages sent shortly before the client posts one more access token to the RS might not successfully reach the destination. Analogously, implementations may want to cancel CoAP observations at the RS registered before the Security Context is replaced, or conversely they will need to implement a mechanism to ensure that those observations are to be protected with the newly derived Security Context.

## 7. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [[I-D.ietf-ace-oauth-authz](#)]. Thus the general security considerations from the framework also apply to this profile.

Furthermore the general security considerations of OSCORE [[RFC8613](#)] also apply to this specific use of the OSCORE protocol.

As previously stated, the proof-of-possession in this profile is performed by both parties verifying that they have established the same Security Context, as specified in [Section 4.3](#), which means that both the OSCORE request and the OSCORE response passes verification. RS authentication requires both that the client trusts the AS and that the OSCORE response from the RS passes verification.

OSCORE is designed to secure point-to-point communication, providing a secure binding between the request and the response(s). Thus the basic OSCORE protocol is not intended for use in point-to-multipoint communication (e.g., multicast, publish-subscribe). Implementers of this profile should make sure that their use case corresponds to the expected use of OSCORE, to prevent weakening the security assurances provided by OSCORE.

Since the use of nonces N1 and N2 during the exchange guarantees uniqueness of AEAD keys and nonces, it is REQUIRED that the exchanged nonces are not reused with the same input keying material even in case of re-boots. This document RECOMMENDS the exchange of 64 bit random nonces. Considering the birthday paradox, the average collision for each nonce will happen after  $2^{32}$  messages, which is considerably more token provisioned than would be expected for

intended applications. If applications use something else, such as a counter, they need to guarantee that reboot and loss of state on either node does not provoke reuse. If that is not guaranteed, nodes are susceptible to reuse of AEAD (nonce, key) pairs, especially since an on-path attacker can cause the use of a previously exchanged client nonce N1 for Security Context establishment by replaying the corresponding client-to-server message.

This profile RECOMMENDS that the RS maintains a single access token for each client. The use of multiple access tokens for a single client increases the strain on the resource server as it must consider every access token and calculate the actual permissions of the client. Also, tokens indicating different or disjoint permissions from each other may lead the server to enforce wrong permissions. If one of the access tokens expires earlier than others, the resulting permissions may offer insufficient protection. Developers SHOULD avoid using multiple access tokens for a same client.

If a single OSCORE Input Material is used with multiple RSs, the RSs can impersonate the client to one of the other RS, and impersonate another RS to the client. If a master secret is used with several clients, the clients can impersonate RS to one of the other clients. Similarly if symmetric keys are used to integrity protect the token between AS and RS and the token can be used with multiple RSs, the RSs can impersonate AS to one of the other RS. If the token key is used for any other communication between the RSs and AS, the RSs can impersonate each other to the AS.

## **8. Privacy Considerations**

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [[I-D.ietf-ace-oauth-authz](#)]. Thus the general privacy considerations from the framework also apply to this profile.

As this document uses OSCORE, thus the privacy considerations from [[RFC8613](#)] apply here as well.

An unprotected response to an unauthorized request may disclose information about the resource server and/or its existing relationship with the client. It is advisable to include as little information as possible in an unencrypted response. When an OSCORE Security Context already exists between the client and the resource server, more detailed information may be included.

The token is sent in the clear to the authz-info endpoint, so if a client uses the same single token from multiple locations with



multiple Resource Servers, it can risk being tracked by the token's value even when the access token is encrypted.

The nonces exchanged in the request and response to the authz-info endpoint are also sent in the clear, so using random nonces is best for privacy (as opposed to, e.g., a counter, that might leak some information about the client).

The identifiers used in OSCORE, negotiated between client and RS are privacy sensitive (see Section 12.8 of [[RFC8613](#)]), and could reveal information about the client, or may be used for correlating requests from one client.

Note that some information might still leak after OSCORE is established, due to observable message sizes, the source, and the destination addresses.

## **9. IANA Considerations**

Note to RFC Editor: Please replace all occurrences of "[[this document]]" with the RFC number of this document. Please add a reference to the IANA ACE Profile registry in the nextt subsection once it has been created by IANA, and then delete this paragraph.

### **9.1. ACE Profile Registry**

The following registration is done for the ACE Profile Registry following the procedure specified in section 8.8 of [[I-D.ietf-ace-oauth-authz](#)]:

- \*Name: coap\_oscore
- \*Description: Profile for using OSCORE to secure communication between constrained nodes using the Authentication and Authorization for Constrained Environments framework.
- \*CBOR Value: TBD (value between 1 and 255)
- \*Reference: [[this document]]

### **9.2. OAuth Parameters Registry**

The following registrations are done for the OAuth Parameters Registry [[IANA.OAuthParameters](#)] following the procedure specified in section 11.2 of [[RFC6749](#)]:

- \*Parameter name: nonce1
- \*Parameter usage location: client-rs request
- \*Change Controller: IESG
- \*Specification Document(s): [[this document]]
- \*Parameter name: nonce2
- \*Parameter usage location: rs-client response
- \*Change Controller: IESG

- \*Specification Document(s): [[this document]]
- \*Parameter name: ace\_client\_recipientid
- \*Parameter usage location: client-rs request
- \*Change Controller: IESG
- \*Specification Document(s): [[this document]]
- \*Parameter name: ace\_server\_recipientid
- \*Parameter usage location: rs-client response
- \*Change Controller: IESG
- \*Specification Document(s): [[this document]]

### 9.3. OAuth Parameters CBOR Mappings Registry

The following registrations are done for the OAuth Parameters CBOR Mappings Registry following the procedure specified in section 8.10 of [\[I-D.ietf-ace-oauth-authz\]](#):

- \*Name: nonce1
- \*CBOR Key: TBD1
- \*Value Type: bstr
- \*Reference: [[this document]]
- \*Name: nonce2
- \*CBOR Key: TBD2
- \*Value Type: bstr
- \*Reference: [[this document]]
- \*Name: ace\_client\_recipientid
- \*CBOR Key: TBD3
- \*Value Type: bstr
- \*Reference: [[this document]]
- \*Name: ace\_server\_recipientid
- \*CBOR Key: TBD4
- \*Value Type: bstr
- \*Reference: [[this document]]

### 9.4. OSCORE Security Context Parameters Registry

It is requested that IANA create a new registry entitled "OSCORE Security Context Parameters" registry. The registry is to be created as Expert Review Required. Guidelines for the experts is provided [Section 9.7](#). It should be noted that in addition to the expert review, some portions of the registry require a specification, potentially on standards track, be supplied as well.

The columns of the registry are:

**name** The JSON name requested (e.g., "ms"). Because a core goal of this document is for the resulting representations to be compact, it is RECOMMENDED that the name be short. This name is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts determine that

there is a compelling reason to allow an exception. The name is not used in the CBOR encoding.

**CBOR label** The value to be used to identify this algorithm. Map key labels MUST be unique. The label can be a positive integer, a negative integer or a string. Integer values between -256 and 255 and strings of length 1 are designated as Standards Track Document required. Integer values from -65536 to -257 and from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values greater than 65535 and strings of length greater than 2 are designated as expert review. Integer values less than -65536 are marked as private use.

**CBOR Type** This field contains the CBOR type for the field.

**registry** This field denotes the registry that values may come from, if one exists.

**description** This field contains a brief description for the field.

**specification** This contains a pointer to the public specification for the field if one exists

This registry will be initially populated by the values in [Table 1](#). The specification column for all of these entries will be this document and [[RFC8613](#)].

## 9.5. CWT Confirmation Methods Registry

The following registration is done for the CWT Confirmation Methods Registry [[IANA.CWTConfirmationMethods](#)] following the procedure specified in section 7.2.1 of [[RFC8747](#)]:

- \*Confirmation Method Name: "osc"
- \*Confirmation Method Description: OSCORE\_Input\_Material carrying the parameters for using OSCORE per-message security with implicit key confirmation
- \*Confirmation Key: TBD (value between 4 and 255)
- \*Confirmation Value Type(s): map
- \*Change Controller: IESG
- \*Specification Document(s): [Section 3.2.1](#) of [[this document]]

## 9.6. JWT Confirmation Methods Registry

The following registration is done for the JWT Confirmation Methods Registry [[IANA.JWTConfirmationMethods](#)] following the procedure specified in section 6.2.1 of [[RFC7800](#)]:

- \*Confirmation Method Value: "osc"
- \*Confirmation Method Description: OSCORE\_Input\_Material carrying the parameters for using OSCORE per-message security with implicit key confirmation
- \*Change Controller: IESG
- \*Specification Document(s): [Section 3.2.1](#) of [[this document]]

## 9.7. Expert Review Instructions

The IANA registry established in this document is defined to use the Expert Review registration policy. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- \*Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments. Code points in other ranges should not be assigned for testing.

- \*Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

- \*Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

## 10. References

### 10.1. Normative References

[COSE.Algorithms] IANA, "COSE Algorithms", <<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.

[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)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-40, 26 April

2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-40.txt>>.

**[I-D.ietf-ace-oauth-params]**

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-14, 25 March 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-14.txt>>.

**[I-D.ietf-cose-rfc8152bis-algs]**

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

**[I-D.ietf-cose-rfc8152bis-struct]**

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

**[IANA.CWTConfirmationMethods]** IANA, "CWT Confirmation Methods", <<https://www.iana.org/assignments/cwt/cwt.xhtml#confirmation-methods>>.

**[IANA.JWTConfirmationMethods]** IANA, "JWT Confirmation Methods", <<https://www.iana.org/assignments/jwt/jwt.xhtml#confirmation-methods>>.

**[IANA.OAuthParameters]** IANA, "OAuth Parameters", <<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#parameters>>.

**[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>>.

**[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>>.

**[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>>.

**[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>>.

**[RFC8392]**

Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

**[RFC8610]**

Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

**[RFC8613]**

Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

**[RFC8949]**

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

## **10.2. Informative References**

**[I-D.ietf-tls-dtls13]** Rescorla, E., Tschofenig, H., and N. Modadugu,

"The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.

**[RFC4949]**

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

**[RFC6749]**

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.

**[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>>.

**[RFC7800]**

Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC

7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.

## Appendix A. Profile Requirements

This section lists the specifications on this profile based on the requirements on the framework, as requested in Appendix C of [[I-D.ietf-ace-oauth-authz](#)].

- \*Optionally define new methods for the client to discover the necessary permissions and AS for accessing a resource, different from the one proposed in: Not specified
- \*Optionally specify new grant types: Not specified
- \*Optionally define the use of client certificates as client credential type: Not specified
- \*Specify the communication protocol the client and RS the must use: CoAP
- \*Specify the security protocol the client and RS must use to protect their communication: OSCORE
- \*Specify how the client and the RS mutually authenticate: Implicitly by possession of a common OSCORE security context. Note that the mutual authentication is not completed before the client has verified an OSCORE response using this security context.
- \*Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of- possession protocol: OSCORE algorithms; pre-established symmetric keys
- \*Specify a unique ace\_profile identifier: coap\_oscore
- \*If introspection is supported: Specify the communication and security protocol for introspection: HTTP/CoAP (+ TLS/DTLS/ OSCORE)
- \*Specify the communication and security protocol for interactions between client and AS: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- \*Specify how/if the authz-info endpoint is protected, including how error responses are protected: Not protected.
- \*Optionally define other methods of token transport than the authz- info endpoint: Not defined

## Acknowledgments

The authors wish to thank Jim Schaad and Marco Tiloca for the substantial input to this document, as well as Elwyn Davies, Linda Dunbar, Roman Danyliw, Martin Duke, Lars Eggert, Murray Kucherawy, and Zaheduzzaman Sarker for their reviews and feedback. Special thanks to the responsible area director Benjamin Kaduk for his extensive review and contributed text. Ludwig Seitz worked on this document as part of the CelticNext projects CyberWI, and CRITISEC with funding from Vinnova. The work on this document has been partly supported also by the H2020 project SIFIS-Home (Grant agreement 952652).

## Authors' Addresses

Francesca Palombini  
Ericsson AB

Email: [francesca.palombini@ericsson.com](mailto:francesca.palombini@ericsson.com)

Ludwig Seitz  
Combitech  
Djaeknegatan 31  
SE-211 35 Malmö  
Sweden

Email: [ludwig.seitz@combitech.com](mailto:ludwig.seitz@combitech.com)

Göran Selander  
Ericsson AB

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

Martin Gunnarsson  
RISE  
Scheelevägen 17  
SE-22370 Lund  
Sweden

Email: [martin.gunnarsson@ri.se](mailto:martin.gunnarsson@ri.se)