Network Working Group
Internet-Draft
Intended status: Informational
Expires: 30 December 2020

C. Bormann
Universität Bremen TZI
28 June 2020

An Authorization Information Format (AIF) for ACE
draft-bormann-core-ace-aif-09

Abstract

Constrained Devices as they are used in the "Internet of Things" need security. One important element of this security is that devices in the Internet of Things need to be able to decide which operations requested of them should be considered authorized, need to ascertain that the authorization to request the operation does apply to the actual requester, and need to ascertain that other devices they place requests on are the ones they intended.

To transfer detailed authorization information from an authorization manager (such as an ACE-OAuth Authorization Server) to a device, a representation format is needed. This document provides a suggestion for such a format, the Authorization Information Format (AIF). AIF is defined both as a general structure that can be used for many different applications and as a specific refinement that describes REST resources and the permissions on them.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Terminology 2
2. Information Model 3
2.1. REST-specific model 4
2.2. Limitations 4
2.3. Extended REST-specific model 5
3. Data Model 5
4. Media Types 8
5. IANA Considerations 8
5.1. Media Types 8
5.2. Registries 8
5.3. Content-Format 8
6. Security Considerations 9
7. References 9
7.1. Normative References 9
7.2. Informative References 9
Acknowledgements 11
Author's Address 11

1. Introduction

(See Abstract.)

1.1. Terminology

This memo uses terms from [RFC7252] and [RFC4949].

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. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

(Note that this document is itself informational, but it is discussing normative statements that MUST be put into concrete terms in each specification that makes use of this document.)

The term "byte", abbreviated by "B", is used in its now customary sense as a synonym for "octet".

2. Information Model

Authorizations are generally expressed through some data structures that are cryptographically secured (or transmitted in a secure way). This section discusses the information model underlying the payload of that data (as opposed to the cryptographic armor around it).

For the purposes of this strawman, the underlying access control model will be that of an access matrix, which gives a set of permissions for each possible combination of a subject and an object. We do not concern the AIF format with the subject for which the AIF object is issued, focusing the AIF object on a single row in the access matrix (such a row traditionally is also called a capability list). As a consequence, AIF MUST be used in a way that the subject of the authorizations is unambiguously identified (e.g., as part of the armor around it).

The generic model of a such a capability list is a list of pairs of object identifiers and the permissions the subject has on the object(s) identified.

```
AIF-Generic<Toid, Tperm> = [* [Toid, Tperm]]
```

Figure 1: Definition of Generic AIF

In a specific data model, the object identifier ("Toid") will often be a text string, and the set of permissions ("Tperm") will be represented by a bitset in turn represented as a number (see Section 3).

```
AIF-Specific = AIF-Generic<tstr, uint>
```

Figure 2: Likely shape of a specific AIF

2.1. REST-specific model

In the specific instantiation of the REST resources and the permissions on them, for the object identifiers ("Toid"), we simply use the URI of a resource on a CoAP server. More specifically, the parts of the URI that identify the server ("authority" in [RFC3986]) are considered the realm of the authentication mechanism (which are handled in the cryptographic armor); we therefore focus on the "path-absolute" and "query" parts of the URI (URI "local-part" in this specification, as expressed by the Uri-Path and Uri-Query options in CoAP). As a consequence, AIF MUST be used in a way that it is unambiguous who is the target (enforcement point) of these authorizations.

For the permissions ("Tperm"), we simplify the model permissions to giving the subset of the CoAP methods permitted. This model is summarized in Table 1.

local-part	Permission Set
/s/light	GET
/a/led	PUT, GET
/dtls	POST

Table 1: An authorization instance in the AIF Information Model

2.2. Limitations

This simple information model only allows granting permissions for statically identifiable objects, e.g. URIs for the REST-specific instantiation. One might be tempted to extend the model towards URI templates [RFC6570], however, that requires some considerations of the ease and unambiguity of matching a given URI against a set of templates in an AIF object.

This simple information model also doesn’t allow further conditionalizing access based on state outside the identification of objects (e.g., "opening a door is allowed if that isn’t locked").

Finally, the model does not provide any special access for a set of resources that are specific to a subject, e.g. that the subject created itself by previous operations (PUT, POST) or that were specifically created for the subject by others.

2.3. Extended REST-specific model

The extended REST-specific model addresses the need to provide defined access to dynamic resources that were created by the subject itself, specifically, a resource that is made known to the subject by providing Location-* options in a CoAP result or using the Location header field in HTTP [RFC7231] (the Location-indicating mechanisms). (The concept is somewhat comparable to "ACL inheritance" in NFSv4 [rfc5661], except that it does not use a containment relationship but the fact that the dynamic resource was created from a resource to which the subject had access.)

local-part	Permission Set
/a/make-coffee	POST, Dynamic-GET, Dynamic-DELETE

Table 2: An authorization instance in the AIF Information Model

For a method X, the presence of a Dynamic-X permission means that the subject holds permission to exercise the method X on resources that have been returned by a Location-indicating mechanism to a request that the subject made to the resource listed ("/a/make-coffee" in the example, which might return the location of a resource that allows GET to find out about the status and DELETE to cancel the coffee-making operation).

Since the use of the extension defined in this section can be detected by the mentioning of the Dynamic-X permissions, there is no need for another explicit switch between the basic and the extended model; the extended model is always presumed once a Dynamic-X permission is present.

3. Data Model

Different data model specializations can be defined for the generic information model given above.

In this section, we will give the data model for basic REST authorization. As discussed, the object identifier is specialized as a text string giving a relative URI (local-part as absolute path on the server serving as enforcement point). The permission set is specialized to a single number by the following steps:

* The entries in the table that specify the same local-part are merged into a single entry that specifies the union of the permission sets.

* The (non-dynamic) methods in the permission sets are converted into their CoAP method numbers, minus 1.

* Dynamic-X permissions are converted into what the number would have been for X, plus a Dynamic-Offset chosen as 32 (e.g., 35 for Dynamic-DELETE).

* The set of numbers is converted into a single number by taking each number to the power of two and computing the inclusive OR of the binary representations of all the power values.

This data model could be interchanged in the JSON [RFC8259] representation given in Figure 3.

```
[["/s/light", 1], ["/a/led", 5], ["/dtls", 2]]
```

Figure 3: An authorization instance encoded in JSON (46 bytes)

In CDDL [RFC8610], a straightforward specification of the data model (including both the methods from [RFC7252] and the new ones from [RFC8132], identified by the method code minus 1) is:

```
   AIF-REST = AIF-Generic<path, permissions>
   path = tstr   ; URI relative to enforcement point
   permissions = uint .bits methods
   methods = &(
     GET: 0
     POST: 1
     PUT: 2
     DELETE: 3
     FETCH: 4
     PATCH: 5
     iPATCH: 6
     Dynamic-GET: 32; 0 .plus Dynamic-Offset
     Dynamic-POST: 33; 1 .plus Dynamic-Offset
     Dynamic-PUT: 34; 2 .plus Dynamic-Offset
     Dynamic-DELETE: 35; 3 .plus Dynamic-Offset
     Dynamic-FETCH: 36; 4 .plus Dynamic-Offset
     Dynamic-PATCH: 37; 5 .plus Dynamic-Offset
     Dynamic-iPATCH: 38; 6 .plus Dynamic-Offset
   )
```

Figure 4: AIF in CDDL

A representation of this information in CBOR [RFC7049] is given in Figure 5; again, several optimizations/improvements are possible.

```
   83                        # array(3)
      82                     # array(2)
         68                  # text(8)
            2f732f6c69676874 # "/s/light"
         01                  # unsigned(1)
      82                     # array(2)
         66                  # text(6)
            2f612f6c6564     # "/a/led"
         05                  # unsigned(5)
      82                     # array(2)
         65                  # text(5)
            2f64746c73       # "/dtls"
         02                  # unsigned(2)
```

Figure 5: An authorization instance encoded in CBOR (29 bytes)

Note that choosing 32 as Dynamic-Offset means that all future CoAP methods that can be registered can be represented both as themselves and in the Dynamic-X variant, but only the dynamic forms of methods 1 to 21 are typically usable in a JSON form [RFC7493].

4. Media Types

This specification defines media types for the generic information model, expressed in JSON ("application/aif+json") or in CBOR ("application/aif+cbor"). These media types have parameters for specifying "Toid" and "Tperm"; default values are the values "local-uri" for "Toid" and "REST-method-set" for "Tperm".

[Insert lots of boilerplate here]

A specification that wants to use Generic AIF with different "Toid" and/or "Tperm" is expected to request these as media type parameters (Section 5.2) and register a corresponding Content-Format (Section 5.3).

5. IANA Considerations

5.1. Media Types

See Section 4.

5.2. Registries

IANA is requested to create a registry for AIF with two sub-registries for "Toid" and "Tperm", populated with:

Subregistry	name	Description/Specification
Toid	local-part	local-part of URI as specified in [RFCthis]
Tperm	REST-method-set	set of REST methods represented as specified in [RFCthis]

Table 3

The registration policy is Specification required [RFC8126]. The designated expert will engage with the submitter to ascertain the requirements of this document are addressed.

5.3. Content-Format

IANA is requested to register Content-Format numbers in the CoRE Parameters Registry [IANA.core-parameters], as follows:

6. Security Considerations

(TBD. Some issues are already discussed in the security considerations of [RFC7252] and in [RFC8576].)

7. References

7.1. 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>.

[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>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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>.

7.2. Informative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-11, 18 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-11.txt>.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-11, 18 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-11.txt>.

[IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <http://www.iana.org/assignments/core-parameters>.

[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>.

[rfc5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <https://www.rfc-editor.org/info/rfc5661>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[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>.

[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>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8576] Garcia-Morchon, O., Kumar, S., and M. Sethi, "Internet of Things (IoT) Security: State of the Art and Challenges", RFC 8576, DOI 10.17487/RFC8576, April 2019, <https://www.rfc-editor.org/info/rfc8576>.

Acknowledgements

Jim Schaad and Francesca Palombini provided comments that shaped the direction of this document.

Author’s Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2021

F. Palombini
Ericsson AB
M. Tiloca
RISE AB

2 November 2020

Key Provisioning for Group Communication using ACE
draft-ietf-ace-key-groupcomm-10

Abstract

This document defines message formats and procedures for requesting and distributing group keying material using the ACE framework, to protect communications between group members.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/ace-wg/ace-key-groupcomm.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Terminology 3
2. Overview 4
3. Authorization to Join a Group 7
  3.1. Authorization Request 8
  3.2. Authorization Response 10
  3.3. Token Post 10
4. Keying Material Provisioning and Group Membership Management 14
  4.1. Interface at the KDC 15
  4.2. Retrieval of Group Names and URIs 33
  4.3. Joining Exchange 34
  4.4. Retrieval of Updated Keying Material 36
  4.5. Requesting a Change of Keying Material 39
  4.6. Retrieval of Public Keys and Roles for Group Members . . 40
  4.7. Update of Public Key 42
  4.8. Retrieval of Group Policies 43
  4.9. Retrieval of Keying Material Version 44
  4.10. Group Leaving Request 45
5. Removal of a Node from the Group 45
6. ACE Groupcomm Parameters 46
7. Security Considerations 48
  7.1. Update of Keying Material 49
  7.2. Block-Wise Considerations 50
8. IANA Considerations 50
  8.1. Media Type Registrations 50
  8.2. CoAP Content-Formats Registry 51
  8.3. OAuth Parameters Registry 51
  8.4. OAuth Parameters CBOR Mappings Registry 52
  8.5. ACE Groupcomm Parameters Registry 52
  8.6. ACE Groupcomm Key Registry 53
  8.7. ACE Groupcomm Profile Registry 53
  8.8. ACE Groupcomm Policy Registry 54
  8.9. Sequence Number Synchronization Method Registry 55
  8.10. Interface Description (if=) Link Target Attribute Values Registry 55
  8.11. Expert Review Instructions 55
9. References 56
  9.1. Normative References 56
  9.2. Informative References 58

Appendix A. Requirements on Application Profiles 60
Appendix B. Document Updates 63
  B.1. Version -04 to -05 63
  B.2. Version -03 to -04 63
  B.3. Version -02 to -03 64
  B.4. Version -01 to -02 64
  B.5. Version -00 to -01 65
Acknowledgments 66
Authors' Addresses 66

1. Introduction

This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to define the message exchanges used to request, distribute and renew the keying material in a group communication scenario, e.g. based on multicast [I-D.ietf-core-groupcomm-bis] or on publishing-subscribing [I-D.ietf-core-coap-pubsub]. The ACE framework is based on CBOR [I-D.ietf-cbor-7049bis], so CBOR is the format used in this specification. However, using JSON [RFC8259] instead of CBOR is possible, using the conversion method specified in Sections 6.1 and 6.2 of [I-D.ietf-cbor-7049bis].

Profiles that use group communication can build on this document, by defining a number of details such as the exact group communication protocol and security protocols used. The specific list of details a profile needs to define is shown in Appendix A.

If the application requires backward and forward security, new keying material is generated and distributed to the group upon membership changes. A key management scheme performs the actual distribution of the new keying material to the group. In particular, the key management scheme rekeys the current group members when a new node joins the group, and the remaining group members when a node leaves the group. Rekeying mechanisms can be based on [RFC2093], [RFC2094] and [RFC2627].

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 [I-D.ietf-ace-oauth-authz][I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], such as Authorization Server (AS) and Resource Server (RS).

This document uses names or identifiers for groups and nodes. Their different meanings are summarized here:

* "Group name" is the invariant once established identifier of the group. It is used in the communication between AS, RS and Client to identify the group.

* "GROUPNAME" is the invariant once established text string used in URIs. GROUPNAME maps to the group name of a group, although it is not necessarily the same.

* "Group identifier" is the identifier of the group keying material. Opposite to group name and GROUPNAME, this identifier changes over time, when the keying material is updated.

* "Node name" is the invariant once established identifier of the node. It is used in the communication between AS, RS and Client to identify a member of the group.

* "NODENAME" is the invariant once established text string used in URIs. NODENAME is used to identify a node in a group.

This document additionally uses the following terminology:

* Transport profile, to indicate a profile of ACE as per Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]. A transport profile specifies the communication protocol and communication security protocol between an ACE Client and Resource Server, as well as proof-of-possession methods, if it supports proof-of-possession access tokens, etc. Tranport profiles of ACE include, for instance, [I-D.ietf-ace-oscore-profile], [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-mqtt-tls-profile].

* Application profile, that defines how applications enforce and use supporting security services they require. These services may include, for instance, provisioning, revocation and distribution of keying material. An application profile may define specific procedures and message formats.

2. Overview

The full procedure can be separated in two phases: the first one follows the ACE framework, between Client, AS and KDC; the second one is the key distribution between Client and KDC. After the two phases are completed, the Client is able to participate in the group communication, via a Dispatcher entity.

```
       +------------+                  +-----------+
       |     AS     |                  |    KDC    |
       |            |        .-------->|           |
       +------------+       /          +-----------+
             ^             /
             |            /
             v           /                           +------------+
       +------------+   /      +------------+        |+------------+
       |   Client   |<-'       | Dispatcher |        ||+------------+
       |            |<-------->|            |<------->||   Group    |
       +------------+          +------------+         +|  members   |
                                                       +------------+
```

Figure 1: Key Distribution Participants

The following participants (see Figure 1) take part in the authorization and key distribution.

* Client (C): node that wants to join the group communication. It can request write and/or read rights.

* Authorization Server (AS): same as AS in the ACE Framework; it enforces access policies, and knows if a node is allowed to join a given group with write and/or read rights.

* Key Distribution Center (KDC): maintains the keying material to protect group communications, and provides it to Clients authorized to join a given group. During the first part of the exchange (Section 3), it takes the role of the RS in the ACE Framework. During the second part (Section 4), which is not based on the ACE Framework, it distributes the keying material. In addition, it provides the latest keying material to group members when requested or, if required by the application, when membership changes.

* Dispatcher: entity through which the Clients communicate with the group and which distributes messages to the group members. Examples of dispatchers are: the Broker node in a pub-sub setting; a relayer node for group communication that delivers group messages as multiple unicast messages to all group members; an implicit entity as in a multicast communication setting, where messages are transmitted to a multicast IP address and delivered on the transport channel.

This document specifies a mechanism for:

* Authorizing a new node to join the group (Section 3), and providing it with the group keying material to communicate with the other group members (Section 4).

* Allowing a group member to leave the group (Section 5).

* Evicting a group member from the group (Section 5).

* Allowing a group member to retrieve keying material (Section 4.4 and Section 4.5).

* Renewing and re-distributing the group keying material (rekeying) upon a membership change in the group (Section 4.10 and Section 5).

Figure 2 provides a high level overview of the message flow for a node joining a group communication setting, which can be expanded as follows.

1. The joining node requests an Access Token from the AS, in order to access a specific group-membership resource on the KDC and hence join the associated group. This exchange between Client and AS MUST be secured, as specified by the transport profile of ACE used between Client and KDC. The joining node will start or continue using a secure communication association with the KDC, according to the response from the AS.

2. The joining node transfers authentication and authorization information to the KDC, by posting the obtained Access Token to the /authz-info endpoint at the KDC. This exchange, and all further communications between the Client and the KDC, MUST occur over the secure channel established as a result of the transport profile of ACE used between Client and KDC. After that, a joining node MUST have a secure communication association established with the KDC, before starting to join a group under that KDC. Possible ways to provide a secure communication association are described in the DTLS transport profile [I-D.ietf-ace-dtls-authorize] and OSCORE transport profile [I-D.ietf-ace-oscore-profile] of ACE.

3. The joining node starts the joining process to become a member of the group, by accessing the related group-membership resource at the KDC. At the end of the joining process, the joining node has received from the KDC the parameters and keying material to securely communicate with the other members of the group, and the KDC has stored the association between the authorization information from the access token and the secure session with the joining node.

4. The joining node and the KDC maintain the secure association, to support possible future communications. These especially include key management operations, such as retrieval of updated keying material or participation to a group rekeying process.

5. The joining node can communicate securely with the other group members, using the keying material provided in step 3.

```
               C                           AS  KDC                Group
               |                           |    |                 Member
             / |                           |    |                     |
             | |   Authorization Request   |    |                     |
    Defined  | |-------------------------->|    |                     |
    in the   | |                           |    |                     |
      ACE    | |   Authorization Response  |    |                     |
   framework | |<--------------------------|    |                     |
             | |                                |                     |
             \ |---------- Token Post --------->|                     |
               |                                |                     |
               |------- Joining Request ------->|                     |
               |                                |                     |
               |<------ Joining Response -------|-- Group Rekeying -->|
               |                                |                     |
               |                                     Dispatcher       |
               |                                         |            |
               |<===== Secure group communication =======|===========>|
               |                                         |            |
```

Figure 2: Message Flow Upon New Node's Joining

3. Authorization to Join a Group

This section describes in detail the format of messages exchanged by the participants when a node requests access to a given group. This exchange is based on ACE [I-D.ietf-ace-oauth-authz].

As defined in [I-D.ietf-ace-oauth-authz], the Client requests from the AS an authorization to join the group through the KDC (see Section 3.1). If the request is approved and authorization is granted, the AS provides the Client with a proof-of-possession access token and parameters to securely communicate with the KDC (see Section 3.2).

Communications between the Client and the AS MUST be secured, as defined by the transport profile of ACE used. The Content-Format used in the message depends on the used transport profile of ACE. For example, this can be application/ace+cbor for the first two messages and application/cwt for the third message, which are defined in the ACE framework. The transport profile of ACE also defines a number of details such as the communication and security protocols used with the KDC (see Appendix C of [I-D.ietf-ace-oauth-authz]).

Figure 3 gives an overview of the exchange described above.

```
      Client                                            AS  KDC
         |                                              |    |
         |---- Authorization Request: POST /token ------>|    |
         |                                              |    |
         |<--- Authorization Response: 2.01 (Created) ---|    |
         |                                              |    |
         |----- POST Token: POST /authz-info --------------->|
         |                                                   |
```

Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

The Authorization Request sent from the Client to the AS is defined in Section 5.6.1 of [I-D.ietf-ace-oauth-authz] and MAY contain the following parameters, which, if included, MUST have the corresponding values:

* 'scope', containing the identifier of the specific group(s), or topic(s) in the case of pub-sub, that the Client wishes to access, and optionally the role(s) that the Client wishes to take.

  This value is a CBOR byte string, wrapping a CBOR array of one or more entries.

  By default, each entry is encoded as specified by [I-D.ietf-ace-aif]. The object identifier Toid corresponds to the group name and MUST be encoded as a tstr. The permission set Tperm indicates the roles that the client wishes to take in the group. It is up to the application profiles to define Tperm (REQ2) and register Toid and Tperm to fit the use case. An example of scope using the AIF format is given in Figure 4.

  Otherwise, each scope entry can be defined as a CBOR array, which contains:

  - As first element, the identifier of the specific group or topic, encoded as a tstr.

  - Optionally, as second element, the role (or CBOR array of roles) that the Client wishes to take in the group. This element is optional since roles may have been pre-assigned to the Client, as associated to its verifiable identity credentials. Alternatively, the application may have defined a single, well-known role for the target resource(s) and audience(s).

In each entry, the encoding of the role identifiers is application specific, and part of the requirements for the application profile (REQ2). In particular, the application profile may specify CBOR values to use for abbreviating role identifiers (OPT7).

An example of CDDL definition [RFC8610] of scope using the format above, with group name and role identifiers encoded as text strings is given in Figure 5.

* 'audience', with an identifier of a KDC.

As defined in [I-D.ietf-ace-oauth-authz], other additional parameters can be included if necessary.

```
gname = tstr

permissions = uint . bits roles

roles = &(
   Requester: 1,
   Responder: 2,
   Monitor: 3,
   Verifier: 4
)

scope_entry = AIF_Generic<gname, permissions>

scope = << [ + scope_entry ] >>
```

Figure 4: Example CDLL definition of scope, using the default Authorization Information Format

```
gname = tstr

role = tstr

scope_entry = [ gname , ? ( role / [ 2*role ] ) ]

scope = << [ + scope_entry ] >>
```

Figure 5: CDLL definition of scope, using as example group name encoded as tstr and role as tstr

3.2. Authorization Response

The Authorization Response sent from the AS to the Client is defined in Section 5.6.2 of [I-D.ietf-ace-oauth-authz]. Note that the parameter 'expires_in' MAY be omitted if the application defines how the expiration time is communicated to the Client via other means, or if it establishes a default value.

Additionally, when included, the following parameter MUST have the corresponding values:

* 'scope' has the same format and encoding of 'scope' in the Authorization Request, defined in Section 3.1. If this parameter is not present, the granted scope is equal to the one requested in Section 3.1}.

The proof-of-possession access token (in 'access_token' above) MUST contain the following parameters:

* a confirmation claim (see for example 'cnf' defined in Section 3.1 of [RFC8747] for CWT);

* an expiration time claim (see for example 'exp' defined in Section 3.1.4 of [RFC8392] for CWT);

* a scope claim (see for example 'scope' registered in Section 8.13 of [I-D.ietf-ace-oauth-authz] for CWT). This claim has the same encoding as the 'scope' parameter above. Additionally, this claim has the same value of the 'scope' parameter if the parameter is present in the message, or it takes the value of 'scope' in the Authorization Request otherwise.

The access token MAY additionally contain other claims that the transport profile of ACE requires, or other optional parameters.

When receiving an Authorization Request from a Client that was previously authorized, and for which the AS still owns a valid non-expired access token, the AS MAY reply with that token. Note that it is up to application profiles of ACE to make sure that re-posting the same token does not cause re-use of keying material between nodes (for example, that is done with the use of random nonces in [I-D.ietf-ace-oscore-profile]).

3.3. Token Post

The Client sends a CoAP POST request including the access token to the KDC, as specified in Section 5.8.1 of [I-D.ietf-ace-oauth-authz].

This request differs from the one defined in [I-D.ietf-ace-oauth-authz], because it allows to transport additional encoding information about the public keys in the group, used for source authentication, as well as any other group parameters. The joining node MAY ask for this information from the KDC in the same message it uses to POST the token to the RS.

The payload of the message MUST be formatted as a CBOR map including the access token.

Additionally, the CoAP POST request MAY contain the following parameter, which, if included, MUST have the corresponding values:

* 'sign_info' defined in Section 3.3.1, encoding the CBOR simple value Null to require information about the signature algorithm, signature algorithm parameters, signature key parameters and on the exact encoding of public keys used in the group.

Alternatively, the joining node may retrieve this information by other means.

After successful verification, the Client is authorized to receive the group keying material from the KDC and join the group.

The KDC replies to the Client with a 2.01 (Created) response, using Content-Format "application/ace+cbor" defined in Section 8.14 of [I-D.ietf-ace-oauth-authz].

The payload of the 2.01 response is a CBOR map. If the access token contains a role that requires the Client to send its own public key to the KDC when joining the group, the CBOR map MUST include the parameter 'kdcchallenge' defined in Section 3.3.2, specifying a dedicated challenge N_S generated by the KDC. The Client uses this challenge to prove possession of its own private key (see the 'client_cred_verify' parameter in Section 4). Note that the payload format of the response deviates from the one defined in the ACE framework (see Section 5.8.1 of [I-D.ietf-ace-oauth-authz]), which has no payload.

The KDC MUST store the 'kdcchallenge' value associated to the Client at least until it receives a join request from it (see Section 4.3), to be able to verify that the Client possesses its own private key. The same challenge MAY be reused several times by the Client, to generate a new proof of possession, e.g. in case of update of the public key, or to join a different group with a different signing key, so it is RECOMMENDED that the KDC keeps storing the 'kdcchallenge' after the first join is processed as well. If the KDC has already discarded the 'kdcchallenge', that will trigger an error response with a newly generated 'kdcchallenge' that the Client can use to restart the join process, as specified in Section 4.3.

If 'sign_info' is included in the request, the KDC MAY include the 'sign_info' parameter defined in Section 3.3.1, with the same encoding. Note that the field 'id' takes the value of the group name for which the 'sign_info_entry' applies to.

Note that the CBOR map specified as payload of the 2.01 (Created) response may include further parameters, e.g. according to the signalled transport profile of ACE. Application profiles MAY define the additional parameters to use within this exchange (OPT2b).

Application profiles of this specification MAY define alternative specific negotiations of parameter values for the signature algorithm and signature keys, if 'sign_info' is not used (OPT2a).

3.3.1. 'sign_info' Parameter

The 'sign_info' parameter is an OPTIONAL parameter of the Token Post response message defined in Section 5.1.2. of [I-D.ietf-ace-oauth-authz]. This parameter contains information and parameters about the signature algorithm and the public keys to be used between the Client and the RS. Its exact content is application specific.

In this specification and in application profiles building on it, this parameter is used to ask and retrieve from the KDC information about the signature algorithm and related parameters used in the group.

When used in the request, the 'sign_info' encodes the CBOR simple value Null, to require information and parameters on the signature algorithm and on the public keys used.

The CDDL notation [RFC8610] of the 'sign_info' parameter formatted as in the request is given below.

```
   sign_info_req = nil
```

The ’sign_info’ parameter of the 2.01 (Created) response is a CBOR array of one or more elements. The number of elements is at most the number of groups that the client has been authorized to join. Each element contains information about signing parameters and keys for one or more group or topic, and is formatted as follows.

* The first element ’id’ is a group name or an array of group names for the group(s) for which this information applies. Below, each specified group name is referred to as ’gname’.

* The second element ’sign_alg’ is an integer or a text string if the POST request included the ’sign_info’ parameter with value Null, and indicates the signature algorithm used in the group(s) identified by (the set of) ’gname’. It is REQUIRED of the application profiles to define specific values that this parameter can take (REQ3), selected from the set of signing algorithms of the COSE Algorithms registry [COSE.Algorithms].

* The third element ’sign_parameters’ is a CBOR array indicating the parameters of the signature algorithm used in the group(s) identified by (the set of) ’gname’. Its content depends on the value of ’sign_alg’. It is REQUIRED of the application profiles to define the possible values and structure for the elements of this parameter (REQ4).

* The fourth element ’sign_key_parameters’ is a CBOR array indicating the parameters of the key used with the signature algorithm, in the group(s) identified by (the set of) ’gname’. Its content depends on the value of ’sign_alg’. It is REQUIRED of the application profiles to define the possible values and structure for the elements of this parameter (REQ5).

* The fifth element ’pub_key_enc’ parameter is either a CBOR integer indicating the encoding of public keys used in the group(s) identified by (the set of) ’gname’, or has value Null indicating that the KDC does not act as repository of public keys for group members. Its acceptable values are taken from the "CWT Confirmation Method" Registry defined in [RFC8747]. It is REQUIRED of the application profiles to define specific values to use for this parameter (REQ6).

The CDDL notation [RFC8610] of the ’sign_info’ parameter formatted as in the response is given below.

```
      sign_info_res = [ + sign_info_entry ]

      sign_info_entry =
      [
        id : gname / [ + gname ],
        sign_alg : int / tstr,
        sign_parameters : [ any ],
        sign_key_parameters : [ any ],
        pub_key_enc = int / nil
      ]

      gname = tstr
```

3.3.2. 'kdcchallenge' Parameter

The 'kdcchallenge' parameter is an OPTIONAL parameter of the Token Post response message defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. This parameter contains a challenge generated by the KDC and provided to the Client. The Client may use this challenge to prove possession of its own private key in the Joining Request (see the 'client_cred_verify' parameter in Section 4).

4. Keying Material Provisioning and Group Membership Management

This section defines the interface available at the KDC. Moreover, this section specifies how the clients can use this interface to join a group, leave a group, retrieve the group policies or the new keying material.

During the first exchange with the KDC ("Joining") after posting the Token, the Client sends a request to the KDC, specifying the group it wishes to join (see Section 4.3). Then, the KDC verifies the access token and that the Client is authorized to join that group. If so, it provides the Client with the keying material to securely communicate with the other members of the group. Whenever used, the Content-Format in messages containing a payload is set to application/ace-groupcomm+cbor, as defined in Section 8.2.

When the Client is already a group member, the Client can use the interface at the KDC to perform the following actions:

* The Client can get the current keying material, for cases such as expiration, loss or suspected mismatch, due to e.g. reboot or missed group rekeying. This is described in Section 4.4.

* The Client can retrieve new keying material for itself. This is described in Section 4.5.

* The Client can get the public keys of other group members. This is described in Section 4.6.

* The Client can upload a new, updated public key at the KDC. This is described in Section 4.7.

* The Client can get the group policies. This is described in Section 4.8.

* The Client can get the version number of the keying material currently used in the group. This is described in Section 4.9.

* The Client can request to leave the group. This is further discussed in Section 4.10.

Upon receiving a request from a Client, the KDC MUST check that it is storing a valid access token from that Client for the group name associated to the endpoint. If that is not the case, i.e. the KDC does not store a valid access token or this is not valid for that Client for the group name, the KDC MUST respond to the Client with a 4.01 (Unauthorized) error message.

4.1. Interface at the KDC

The KDC is configured with the following resources. Note that the root url-path "ace-group" given here are default names: implementations are not required to use these names, and can define their own instead. Each application profile of this specification MUST register a Resource Type for the root url-path (REQ7a), and that Resource Type can be used to discover the correct url to access at the KDC. This Resource Type can also be used at the GROUPNAME sub-resource, to indicate different application profiles for different groups. The Interface Description (if=) Link Target Attribute value ace.group is registered (Section 8.10) and can be used to describe this interface.

* /ace-group: this resource is invariant once established and indicates that this specification is used. If other applications run on a KDC implementing this specification and use this same resource, these applications will collide, and a mechanism will be needed to differentiate the endpoints. This resource supports the FETCH method.

* /ace-group/GROUPNAME: one sub-resource to /ace-group is implemented for each group the KDC manages.

If the value of the GROUPNAME URI path and the group name in the access token scope ('gname' in Section 3.2) do not match, the KDC MUST implement a mechanism to map the GROUPNAME value in the URI to the group name, in order to retrieve the right group (REQ1). Each resource contains the symmetric group keying material for that group. These resources support the GET and POST methods.

* /ace-group/GROUPNAME/pub-key: this resource is invariant once established and contains the public keys of all group members. This resource supports the GET and FETCH methods.

* /ace-group/GROUPNAME/policies: this resource is invariant once established and contains the group policies. This resource supports the GET method.

* /ace-group/GROUPNAME/num: this resource is invariant once established and contains the version number for the symmetric group keying material. This sub-resource supports the GET method.

* /ace-group/GROUPNAME/nodes/NODENAME: one sub-resource to /ace-group/GROUPNAME is implemented for each node in the group the KDC manages. These resources are identified by the node name (in this example, the node name has value "NODENAME"). Each resource contains the group and individual keying material for that node. These resources support the GET, PUT and DELETE methods.

* /ace-group/GROUPNAME/nodes/NODENAME/pub-key: one sub-resource to /ace-group/GROUPNAME/nodes/NODENAME is implemented for each node in the group the KDC manages. These resources are identified by the node name (in this example, the node name has value "NODENAME"). Each resource contains the individual public keying material for that node. These resources support the POST method.

It is REQUIRED of the application profiles of this specification to define what operations (i.e. CoAP methods) are allowed on each resource, for each role defined in Section 3.1 according to REQ2 (REQ7aa).

The details for the handlers of each resource are given in the following sections. These endpoints are used to perform the operations introduced in Section 4.

4.1.1. ace-group

This resource implements a FETCH handler.

4.1.1.1. FETCH Handler

The FETCH handler receives group identifiers and returns the corresponding group names and GROUPNAME URIs.

The handler expects a request with payload formatted as a CBOR map. The payload of this request is a CBOR Map that MUST contain the following fields:

* 'gid', whose value is encoded as a CBOR array, containing one or more group identifiers. The exact encoding of group identifier MUST be specified by the application profile (REQ7b). The Client indicates that it wishes to receive the group names and GROUPNAMEs of all groups having these identifiers.

The handler identifies the groups that are secured by the keying material identified by those group identifiers.

Then, the handler returns a 2.05 (Content) message response with payload formatted as a CBOR map that MUST contain the following fields:

* 'gid', whose value is encoded as a CBOR array, containing zero or more group identifiers. The handler indicates that those are the identifiers it is sending group names and GROUPNAMEs for. This CBOR array is a subset of the 'gid' array in the FETCH request.

* 'gname', whose value is encoded as a CBOR array, containing zero or more group names. The elements of this array are encoded as text strings. Each element of index i of this CBOR array corresponds to the element of group identifier i in the 'gid' array.

* 'guri', whose value is encoded as a CBOR array, containing zero or more URIs, each indicating a GROUPNAME resource. The elements of this array are encoded as text strings. Each element of index i of this CBOR array corresponds to the element of group identifier i in the 'gid' array.

If the KDC does not find any group associated with the specified group identifiers, the handler returns a response with payload formatted as a CBOR byte string of zero length.

Note that the KDC only verifies that the node is authorized by the AS to access this resource. Nodes that are not members of the group but are authorized to do signature verifications on the group messages may be allowed to access this resource, if the application needs it.

4.1.2. ace-group/GROUPNAME

This resource implements GET and POST handlers.

4.1.2.1. POST Handler

The POST handler adds the public key of the client to the list of the group members' public keys and returns the symmetric group keying material for the group identified by "GROUPNAME". Note that the group joining exchange is done by the client via this operation, as described in Section 4.3.

The handler expects a request with payload formatted as a CBOR map which MAY contain the following fields, which, if included, MUST have the corresponding values:

* 'scope', with value the specific resource at the KDC that the Client is authorized to access, i.e. group or topic name, and role(s). This value is a CBOR byte string wrapping one scope entry, as defined in Section 3.1.

* 'get_pub_keys', if the Client wishes to receive the public keys of the other nodes in the group from the KDC. This parameter may be present if the KDC stores the public keys of the nodes in the group and distributes them to the Client; it is useless to have here if the set of public keys of the members of the group is known in another way, e.g. it was provided by the AS. Note that including this parameter may result in a large message size for the following response, which can be inconvenient for resource-constrained devices. The parameter's value is either the CBOR simple value Null or a non-empty CBOR array containing two CBOR arrays:

  - The first array is non-empty. Each element of the first array contains one role or a combination of roles for the group identified by "GROUPNAME". The Client indicates that it wishes to receive the public keys of all group members having any of the single roles, or at least all of the roles indicated in any combinations of roles. For example, the array ["role1", "role2+role3"] indicates that the Client wishes to receive the public keys of all group members that have at least "role1" or at least both "role2" and "role3".

  - The second array is empty.

  If the Client wishes to receive all public keys of all group members, it encodes the 'get_pub_key' parameter as the CBOR simple value Null.

The CDDL definition [RFC8610] of 'get_pub_keys' is given in Figure 6, using as example encoding: node identifier encoded as a CBOR byte string; role identifier encoded as a CBOR text string, and combination of roles encoded as a CBOR array of roles.

Note that the second array (array of node identifiers) is empty for this handler, because the joining node is not expected to filter based on node identifiers, but is not necessarily empty for the value of 'get_pub_keys' received by the handler of FETCH to ace-group/GROUPNAME/pub-key (see Section 4.1.3.1).

Also note that the second array (array of roles) is non-empty for this handler, but that is not necessarily the case for other handlers using this parameter: if this array is empty it means that the client is not filtering public keys based on roles.

Finally, 'get_pub_keys' is never used as an array containing two empty arrays (in CBOR diagnostic notation: [[ ], [ ]]), so if this parameter is received as formatted in that way, it has to be considered malformed.

```
id = bstr

role = tstr

comb_role = [ 2*role ]

get_pub_keys = null / [ [ *(role / comb_role) ], [ *id ] ]
```

Figure 6: CDLL definition of get_pub_keys, using as example node identifier encoded as bstr and role as tstr

* 'client_cred', with value the public key or certificate of the Client, encoded as a CBOR byte string. This field contains the public key of the Client. This field is used if the KDC is managing (collecting from/distributing to the Client) the public keys of the group members, and if the Client's role in the group will require for it to send messages to one or more group members. The default encoding for public keys is COSE Keys. Alternative specific encodings of this parameter MAY be defined in applications of this specification (OPT1 in Appendix A).

* 'cnonce', encoded as a CBOR byte string, and including a dedicated nonce N_C generated by the Client. This parameter MUST be present if the 'client_cred' parameter is present.

* 'client_cred_verify', encoded as a CBOR byte string. This parameter MUST be present if the 'client_cred' parameter is present and no public key associated to the client's token can be retrieved for that group. This parameter contains a signature computed by the Client over the following signature input: the scope (encoded as CBOR byte string), concatenated with N_S (encoded as CBOR byte string) concatenated with N_C (encoded as CBOR byte string), where:

  - scope is the CBOR byte string either specified in the 'scope' parameter above, if present, or as a default scope that the handler is expected to understand, if omitted.

  - N_S is the challenge received from the KDC in the 'kdcchallenge' parameter of the 2.01 (Created) response to the token POST request (see Section 3.3), encoded as a CBOR byte string.

  - N_C is the nonce generated by the Client and specified in the 'cnonce' parameter above, encoded as a CBOR byte string.

  An example of signature input construction to compute 'client_cred_verify' using CBOR encoding is given in Figure 7.

  If the token was not posted (e.g. if it is used directly to validate TLS instead), it is REQUIRED of the specific profile to define how the challenge N_S is generated (REQ17). The Client computes the signature by using its own private key, whose corresponding public key is either directly specified in the 'client_cred' parameter or included in the certificate specified in the 'client_cred' parameter.

* 'pub_keys_repos', can be present if a certificate is present in the 'client_cred' field, with value the URI of the certificate of the Client. This parameter is encoded as a CBOR text string. Alternative specific encodings of this parameter MAY be defined in applications of this specification (OPT3).

* 'control_path', with value a full URI, encoded as a CBOR text string. If 'control_path' is supported by the Client, the Client acts as a CoAP server and hosts a resource at this specific URI. The KDC MAY use this URI to send CoAP requests to the Client (acting as CoAP server in this exchange), for example for individual provisioning of new keying material when performing a group rekeying (see Section 4.4), or to inform the Client of its removal from the group Section 5. If the KDC does not implement mechanisms using this resource, it can just ignore this parameter. Other additional functionalities of this resource MAY be defined

      in application profiles of this specifications (OPT9). In
      particular, this resource is intended for communications
      concerning exclusively the group or topic specified in the 'scope'
      parameter.

```
scope, N_S, and N_C expressed in CBOR diagnostic notation:
      scope = h'826667726F7570316673656E646572'
      N_S = h'018a278f7faab55a'
      N_C = h'25a8991cd700ac01'


scope, N_S, and N_C  as CBOR encoded byte strings:
      scope = 0x4f826667726F7570316673656E646572
      N_S = 0x48018a278f7faab55a
      N_C = 0x4825a8991cd700ac01

input to client_cred_verify signature =
  0x4f 826667726F7570316673656E646572 48 018a278f7faab55a 48 25a8991cd700ac01
```

Figure 7: Example of signature input construction to compute 'client_cred_verify' using CBOR encoding

The handler extracts the granted scope from the access token, and checks the requested one against the token one. If the requested one is not a subset of the token one, the KDC MUST respond with a 4.01 (Unauthorized) error message. If this join message does not include a 'scope' field, the KDC is expected to understand which group and role(s) the Client is requesting (e.g. there is only one the Client has been granted). If the KDC can not recognize which scope the Client is requesting, it MUST respond with a 4.00 (Bad Request) error message.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. The KDC MAY respond with an AS Request Creation Hints, as defined in Section 5.1.2 of [I-D.ietf-ace-oauth-authz]. Note that in this case, the content format MUST be set to application/ace+cbor.

If the request is not formatted correctly (i.e. required fields non received or received with incorrect format), the handler MUST respond with a 4.00 (Bad Request) error message. The response MAY contain a CBOR map in the payload with content format application/ace+cbor, e.g. it could send back 'sign_info_res' with 'pub_key_enc' set to Null if the Client sent its own public key and the KDC is not set to store public keys of the group members. If the request contained unknown or non-expected fields present, the handler MUST silently drop them and continue processing. Application profiles MAY define optional or mandatory payload formats for specific error cases (OPT6).

If the KDC stores the group members' public keys, the handler checks if one is included in the 'client_cred' field, retrieves it and associates it to the access token received, after verifications succeeded. In particular, the KDC verifies:

* that such public key has an acceptable format for the group identified by "GROUPNAME", i.e. it is encoded as expected and is compatible with the signature algorithm and possible associated parameters.

* that the signature contained in "client_cred_verify" passes verification.

If that cannot be verified, it is RECOMMENDED that the handler stops the process and responds with a 4.00 (Bad Request) error message. Applications profiles MAY define alternatives (OPT5).

If one public key is already associated to the access token and to that group, but the 'client_cred' is populated with a different public key, the handler MUST delete the previous one and replace it with this one, after verifying the points above.

If no public key is included in the 'client_cred' field, the handler checks if one public key is already associated to the access token received (see Section 4.3 for an example) and to the group identified by "GROUPNAME". If that is not the case, the handler responds with a 4.00 Bad Request error response.

If the token was posted but the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST respond with a 4.00 Bad Request error response, including a newly generated 'kdcchallenge' in a CBOR map in the payload. This error response MUST also have Content-Format application/ace+cbor.

If all verifications succeed, the handler:

* Adds the node to the list of current members of the group.

* Assigns a name NODENAME to the node, and creates a sub-resource to /ace-group/GROUPNAME/ at the KDC (e.g. "/ace-group/GROUPNAME/nodes/NODENAME").

* Associates the identifier "NODENAME" with the access token and the secure session for that node.

* If the KDC manages public keys for group members:

  - Adds the retrieved public key of the node to the list of public keys stored for the group identified by "GROUPNAME"

  - Associates the node's public key with its access token and the group identified by "GROUPNAME", if such association did not already exist.

* Returns a 2.01 (Created) message containing the symmetric group keying material, the group policies and all the public keys of the current members of the group, if the KDC manages those and the Client requested them.

The response message also contains the URI path to the sub-resource created for that node in a Location-Path CoAP option. The payload of the response is formatted as a CBOR map which MUST contain the following fields and values:

* 'gkty', identifying the key type of the 'key' parameter. The set of values can be found in the "Key Type" column of the "ACE Groupcomm Key" Registry. Implementations MUST verify that the key type matches the application profile being used, if present, as registered in the "ACE Groupcomm Key" registry.

* 'key', containing the keying material for the group communication, or information required to derive it.

* 'num', containing the version number of the keying material for the group communication, formatted as an integer. This is a strictly monotonic increasing field. The application profile MUST define the initial version number (REQ19).

The exact format of the 'key' value MUST be defined in applications of this specification (REQ7), as well as values of 'gkty' accepted by the application (REQ8). Additionally, documents specifying the key format MUST register it in the "ACE Groupcomm Key" registry defined in Section 8.6, including its name, type and application profile to be used with.

Name	Key Type Value	Profile	Description
Reserved	0		This value is reserved

Figure 8: Key Type Values

The response SHOULD contain the following parameter:

* 'exp', with value the expiration time of the keying material for the group communication, encoded as a CBOR unsigned integer. This field contains a numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds, analogous to what specified for NumericDate in Section 2 of [RFC7519]. Group members MUST stop using the keying material to protect outgoing messages and retrieve new keying material at the time indicated in this field.

Optionally, the response MAY contain the following parameters, which, if included, MUST have the corresponding values:

* 'ace-groupcomm-profile', with value a CBOR integer that MUST be used to uniquely identify the application profile for group communication. Applications of this specification MUST register an application profile identifier and the related value for this parameter in the "ACE Groupcomm Profile" Registry (REQ12).

* 'pub_keys', may only be present if 'get_pub_keys' was present in the request. This parameter is a CBOR byte string, which encodes the public keys of all the group members paired with the respective member identifiers. The default encoding for public keys is COSE Keys, so the default encoding for 'pub_keys' is a CBOR byte string wrapping a COSE_KeySet (see [I-D.ietf-cose-rfc8152bis-struct]), which contains the public keys of all the members of the group. In particular, each COSE Key in the COSE_KeySet includes the node identifier of the corresponding group member as value of its 'kid' key parameter. Alternative specific encodings of this parameter MAY be defined in applications of this specification (OPT1). The specific format of the node identifiers of group members MUST be specified in the application profile (REQ9).

* 'peer_roles', MUST be present if 'pub_keys' is present. This parameter is a CBOR array of n elements, with n the number of public keys included in the 'pub_keys' parameter (at most the number of members in the group). The i-th element of the array specifies the role (or CBOR array of roles) that the group member associated to the i-th public key in 'pub_keys' has in the group. In particular, each array element is encoded as the role element of a scope entry, as defined in Section 3.1.

* 'group_policies', with value a CBOR map, whose entries specify how the group handles specific management aspects. These include, for instance, approaches to achieve synchronization of sequence numbers among group members. The elements of this field are registered in the "ACE Groupcomm Policy" Registry. This specification defines the three elements "Sequence Number Synchronization Method", "Key Update Check Interval" and "Expiration Delta", which are summarized in Figure 9. Application profiles that build on this document MUST specify the exact content format and default value of included map entries (REQ14).

Name	CBOR label	CBOR type	Description	Reference
Sequence Number Synchroniza-tion Method	TBD1	tstr/int	Method for a re-cipient node to synchronize with sequence numbers of a sender node. Its value is taken from the 'Value' column of the Sequence Number Synchronization Method registry	[[this document]]
Key Update Check Interval	TBD2	int	Polling interval in seconds, to check for new keying material at the KDC	[[this document]]
Expiration Delta	TBD3	uint	Number of seconds from 'exp' until the specified UTC date/time after which group members MUST stop using the keying material to verify incoming messages.	[[this document]]

Figure 9: ACE Groupcomm Policies

* 'mgt_key_material', encoded as a CBOR byte string and containing the administrative keying material to participate in the group rekeying performed by the KDC. The application profile MUST define if this field is used, and if used then MUST specify the exact format and content which depend on the specific rekeying scheme used in the group. If the usage of 'mgt_key_material' is indicated and its format defined for a specific key management scheme, that format must explicitly indicate the key management scheme itself. If a new rekeying scheme is defined to be used for an existing 'mgt_key_material' in an existing profile, then that profile will have to be updated accordingly, especially with respect to the usage of 'mgt_key_material' related format and content (REQ18).

Specific application profiles that build on this document MUST specify the communication protocol that members of the group use to communicate with each other (REQ10) and how exactly the keying material is used to protect the group communication (REQ11).

CBOR labels for these fields are defined in Section 6.

4.1.2.2. GET Handler

The GET handler returns the symmetric group keying material for the group identified by "GROUPNAME".

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. The KDC MAY respond with an AS Request Creation Hints, as defined in Section 5.1.2 of [I-D.ietf-ace-oauth-authz]. Note that in this case the content format MUST be set to application/ace+cbor.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing the symmetric group keying material. The payload of the response is formatted as a CBOR map which MUST contain the parameters 'gkty', 'key' and 'num' specified in Section 4.1.2.1.

The payload MAY also include the parameters 'ace-groupcomm-profile', 'exp', and 'mgt_key_material' parameters specified in Section 4.1.2.1.

4.1.3. ace-group/GROUPNAME/pub-key

If the KDC does not maintain public keys for the group, the handler for any request on this resource returns a 4.05 (Method Not Allowed) error message. If it does, the rest of this section applies.

This resource implements GET and FETCH handlers.

4.1.3.1. FETCH Handler

The FETCH handler receives identifiers of group members for the group identified by "GROUPNAME" and returns the public keys of such group members.

The handler expects a request with payload formatted as a CBOR map, that MUST contain the following fields:

* 'get_pub_keys', whose value is encoded as in Section 4.1.2.1 with the following modification:

  - The first array may be empty, if the Client does not wish to filter the requested public keys based on roles of group members.

  - The second array contains zero or more node identifiers of group members, for the group identified by "GROUPNAME". The Client indicates that it wishes to receive the public keys of all nodes having these node identifiers.

  As mentioned, both arrays can not be empty at the same time.

The specific format of public keys as well as identifiers, roles and combination of roles of group members MUST be specified by the application profile (OPT1, REQ2, REQ9).

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler identifies the public keys of the current group members for which either:

* the role identifier matches with one of those indicated in the request; note that the request can contain a "combination of roles", where the handler select all group members who have all roles included in the combination.

* the node identifier matches with one of those indicated in the request.

Then, the handler returns a 2.05 (Content) message response with payload formatted as a CBOR map, containing only the 'pub_keys' and 'peer_roles' parameters from Section 4.1.2.1. In particular, 'pub_keys' encodes the list of public keys of those group members including the respective member identifiers, while 'peer_roles' encodes their respective role (or CBOR array of roles) in the group. The specific format of public keys as well as of node identifiers of group members is specified by the application profile (OPT1, REQ9).

If the KDC does not store any public key associated with the specified node identifiers, the handler returns a response with payload formatted as a CBOR byte string of zero length.

The handler MAY enforce one of the following policies, in order to handle possible node identifiers that are included in the 'get_pub_keys' parameter of the request but are not associated to any current group member. Such a policy MUST be specified by the application profile (REQ13).

* The KDC silently ignores those node identifiers.

* The KDC retains public keys of group members for a given amount of time after their leaving, before discarding them. As long as such public keys are retained, the KDC provides them to a requesting Client.

Note that this resource handler only verifies that the node is authorized by the AS to access this resource. Nodes that are not members of the group but are authorized to do signature verifications on the group messages may be allowed to access this resource, if the application needs it.

4.1.3.2. GET Handler

The handler expects a GET request.

The KDC performs the same verifications as the FETCH handler in Section 4.1.3.1, and if successful returns the same response as in Section 4.1.3.1 but without filtering based on roles or node identifiers: all the group members' public keys are returned.

Note that this resource handler, as the FETCH handler for the same resource, only verifies that the node is authorized by the AS to access this resource. Nodes that are not members of the group but are authorized to do signature verifications on the group messages may be allowed to access this resource, if the application needs it.

4.1.4. ace-group/GROUPNAME/policies

This resource implements a GET handler.

4.1.4.1. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing the list of policies for the group identified by "GROUPNAME". The payload of the response is formatted as a CBOR map including only the parameter 'group_policies' defined in Section 4.1.2.1 and specifying the current policies in the group. If the KDC does not store any policy, the payload is formatted as a zero-length CBOR byte string.

The specific format and meaning of group policies MUST be specified in the application profile (REQ14).

4.1.5. ace-group/GROUPNAME/num

This resource implements a GET handler.

4.1.5.1. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing an integer that represents the version number of the symmetric group keying material. This number is incremented on the KDC every time the KDC updates the symmetric group keying material, before the new keying material is distributed. This number is stored in persistent storage.

The payload of the response is formatted as a CBOR integer.

4.1.6. ace-group/GROUPNAME/nodes/NODENAME

This resource implements GET, PUT and DELETE handlers.

4.1.6.1. PUT Handler

The PUT handler is used to get the KDC to produce and return individual keying material to protect outgoing messages for the node (identified by "NODENAME") for the group identified by "GROUPNAME". Application profiles MAY also use this handler to rekey the whole group. It is up to the application profiles to specify if this handler supports renewal of individual keying material, renewal of the group keying material or both (OPT8).

The handler expects a request with empty payload.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client, identified by "NODENAME". The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing newly-generated keying material for the Client, and/or, if the application profiles requires it (OPT8), starts the complete group rekeying. The payload of the response is formatted as a CBOR map. The specific format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ15) and registered in Section 8.5.

4.1.6.2. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client, identified by "NODENAME". The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

The handler also verifies that the node sending the request and the node name used in the Uri-Path match. If that is not the case, the handler responds with a 4.01 (Unauthorized) error response.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing both the group keying material and the individual keying material for the Client, or information enabling the Client to derive it. The payload of the response is formatted as a CBOR map. The format for the group keying material is the same as defined in the response of Section 4.1.2.2. The specific format of individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ15) and registered in Section 8.5.

4.1.6.3. DELETE Handler

The DELETE handler removes the node identified by "NODENAME" from the group identified by "GROUPNAME".

The handler expects a request with method DELETE (and empty payload).

The handler verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client, identified by "NODENAME". The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

The handler also verifies that the node sending the request and the node name used in the Uri-Path match. If that is not the case, the handler responds with a 4.01 (Unauthorized) error response.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler removes the client from the group identified by "GROUPNAME", for specific roles if roles were specified in the 'scope' field, or for all roles. That includes removing the public key of the client if the KDC keep tracks of that. Then, the handler delete the sub-resource nodes/NODENAME and returns a 2.02 (Deleted) message with empty payload.

4.1.7. ace-group/GROUPNAME/nodes/NODENAME/pub-key

This resource implements a POST handler, if the KDC stores the public key of group members. If the KDC does not store the public keys of group members, the handler does not implement any method, and every request returns a 4.05 Method Not Allowed error.

4.1.7.1. POST Handler

The POST handler is used to replace the stored public key of this client (identified by "NODENAME") with the one specified in the request at the KDC, for the group identified by "GROUPNAME".

The handler expects a POST request with payload as specified in Section 4.1.2.1, with the difference that it includes only the parameters 'client_cred', 'cnonce' and 'client_cred_verify'. In particular, the signature included in 'client_cred_verify' is expected to be computed as defined in Section 4.1.2.1, with a newly generated N_C nonce and the previously received N_S. The specific format of public keys is specified by the application profile (OPT1).

The handler verifies that the group name GROUPNAME is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If the request is not formatted correctly (i.e. required fields non received or received with incorrect format), the handler MUST respond with a 4.00 (Bad Request) error message. If the request contains unknown or non-expected fields present, the handler MUST silently ignore them and continue processing. Application profiles MAY define optional or mandatory payload formats for specific error cases (OPT6).

Otherwise, the handler checks that the public key specified in the 'client_cred' field has a valid format for the group identified by "GROUPNAME", i.e. it is encoded as expected and is compatible with the signature algorithm and possible associated parameters. If that cannot be successfully verified, the handler MUST respond with a 4.00 (Bad Request) error message. Applications profiles MAY define alternatives (OPT5).

Otherwise, the handler verifies the signature contained in the 'client_cred_verify' field of the request, using the public key specified in the 'client_cred' field. If the signature does not pass verification, the handler MUST respond with a 4.01 (Unauthorized) error message. If the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST respond with a 4.00 Bad Request error response, whose payload is a CBOR map including a newly generated 'kdcchallenge'. This error response MUST also have Content-Format application/ace+cbor.

If verification succeeds, the handler replaces the old public key of the node NODENAME with the one specified in the 'client_cred' field of the request, and stores it as the new current public key of the node NODENAME, in the list of group members' public keys for the group identified by GROUPNAME. Then, the handler replies with a 2.04 (Changed) response, which does not include a payload.

4.2. Retrieval of Group Names and URIs

In case the joining node only knows the group identifier of the group it wishes to join or about which it wishes to get update information from the KDC, the node can contact the KDC to request the corresponding group name and joining resource URI. The node can request several group identifiers at once. It does so by sending a CoAP FETCH request to the /ace-group endpoint at the KDC formatted as defined in Section 4.1.1.1.

Figure 10 gives an overview of the exchanges described above, and Figure 11 shows an example.

```
      Client                                                 KDC
         |                                                    |
         |-------- Group Name and URI Retrieval Request: -------->|
         |                   FETCH /ace-group                 |
         |                                                    |
         |<-Group Name and URI Retrieval Response: 2.05 (Content)-|
         |                                                    |
```

Figure 10: Message Flow of Group Name and URI Retrieval Request-Response

```
        Request:

        Header: FETCH (Code=0.05)
        Uri-Host: "kdc.example.com"
        Uri-Path: "ace-group"
        Content-Format: "application/ace-groupcomm+cbor"
        Payload (in CBOR diagnostic notation):
          { "gid": [01, 02] }

        Response:

        Header: Content (Code=2.05)
        Content-Format: "application/ace-groupcomm+cbor"
        Payload (in CBOR diagnostic notation):
          { "gid": [01, 02], "gname": ["group1", "group2"],
            "guri": ["kdc.example.com/g1", "kdc.example.com/g2"] }
```

Figure 11: Example of Group Name and URI Retrieval Request-Response

4.3. Joining Exchange

Figure 12 gives an overview of the Joining exchange between Client and KDC, when the Client first joins a group, while Figure 13 shows an example.

```
      Client                                                 KDC
         |                                                    |
         |----- Joining Request: POST /ace-group/GROUPNAME ------>|
         |                                                    |
         |<--------- Joining Response: 2.01 (Created) -----------  |
         | Location-Path = "/ace-group/GROUPNAME/nodes/NODENAME"  |
```

Figure 12: Message Flow of First Exchange for Group Joining

```
      Request:

      Header: POST (Code=0.02)
      Uri-Host: "kdc.example.com"
      Uri-Path: "ace-group"
      Uri-Path: "g1"
      Content-Format: "application/ace-groupcomm+cbor"
      Payload (in CBOR diagnostic notation,
               with PUB_KEY and SIG being CBOR byte strings):
        { "scope": << [ "group1", ["sender", "receiver"] ] >> ,
          "get_pub_keys": [["sender"], []], "client_cred": PUB_KEY
          "cnonce": h'6df49c495409a9b5', "client_cred_verify": SIG }

      Response:

      Header: Created (Code=2.01)
      Content-Format: "application/ace-groupcomm+cbor"
      Location-Path: "kdc.example.com"
      Location-Path: "g1"
      Location-Path: "nodes"
      Location-Path: "c101"
      Payload (in CBOR diagnostic notation,
               with KEY being a CBOR byte strings):
        { "gkty": 13, "key": KEY, "num": 12, "exp": 1609459200,
          "pub_keys": << [ PUB_KEY1, PUB_KEY2 ] >>,
          "peer_roles": ["sender", ["sender", "receiver"]] }
```

Figure 13: Example of First Exchange for Group Joining

If not previously established, the Client and the KDC MUST first establish a pairwise secure communication channel (REQ16). This can be achieved, for instance, by using a transport profile of ACE. The Joining exchange MUST occur over that secure channel. The Client and the KDC MAY use that same secure channel to protect further pairwise communications that must be secured.

The secure communication protocol is REQUIRED to establish the secure channel between Client and KDC by using the proof-of-possession key bound to the access token. As a result, the proof-of-possession to bind the access token to the Client is performed by using the proof-of-possession key bound to the access token for establishing secure communication between the Client and the KDC.

To join the group, the Client sends a CoAP POST request to the /ace-group/GROUPNAME endpoint at the KDC, where GROUPNAME is the group name of the group to join, formatted as specified in Section 4.1.2.1. This group name is the same as in the scope entry corresponding to that group, specified in the 'scope' parameter of the Authorization

Request/Response, or it can be retrieved from it. Note that, in case of successful joining, the Client will receive the URI to retrieve group keying material and to leave the group in the Location-Path option of the response.

If the node is joining a group for the first time, and the KDC maintains the public keys of the group members, the Client is REQUIRED to send its own public key and proof of possession ("client_cred" and "client_cred_verify" in Section 4.1.2.1). The request is only accepted if both public key and proof of possession are provided. If a node re-joins a group with the same access token and the same public key, it can omit to send the public key and the proof of possession, or just omit the proof of possession, and the KDC will be able to retrieve its public key associated to its token for that group (if the key has been discarded, the KDC will reply with 4.00 Bad Request, as specified in Section 4.1.2.1). If a node re-joins a group but wants to update its own public key, it needs to send both public key and proof of possession.

If the application requires backward security, the KDC MUST generate new group keying material and securely distribute it to all the current group members, upon a new node's joining the group. To this end, the KDC uses the message format of the response defined in Section 4.1.2.2. Application profiles may define alternative ways of retrieving the keying material, such as sending separate requests to different resources at the KDC (Section 4.1.2.2, Section 4.1.3.2, Section 4.1.4.1). After distributing the new group keying material, the KDC MUST increment the version number of the keying material.

4.4. Retrieval of Updated Keying Material

When any of the following happens, a node MUST stop using the owned group keying material to protect outgoing messages, and SHOULD stop using it to decrypt and verify incoming messages.

* Upon expiration of the keying material, according to what indicated by the KDC with the 'exp' parameter in a Joining Response, or to a pre-configured value.

* Upon receiving a notification of revoked/renewed keying material from the KDC, possibly as part of an update of the keying material (rekeying) triggered by the KDC.

* Upon receiving messages from other group members without being able to retrieve the keying material to correctly decrypt them. This may be due to rekeying messages previously sent by the KDC, that the Client was not able to receive or decrypt.

In either case, if it wants to continue participating in the group communication, the node has to request the latest keying material from the KDC. To this end, the Client sends a CoAP GET request to the /ace-group/GROUPNAME/nodes/NODENAME endpoint at the KDC, formatted as specified in Section 4.1.6.2.

Note that policies can be set up, so that the Client sends a Key Re-Distribution request to the KDC only after a given number of received messages could not be decrypted (because of failed decryption processing or inability to retrieve the necessary keying material).

It is application dependent and pertaining to the particular message exchange (e.g. [I-D.ietf-core-oscore-groupcomm]) to set up these policies for instructing clients to retain incoming messages and for how long (OPT4). This allows clients to possibly decrypt such messages after getting updated keying material, rather than just consider them non valid messages to discard right away.

The same Key Distribution Request could also be sent by the Client without being triggered by a failed decryption of a message, if the Client wants to be sure that it has the latest group keying material. If that is the case, the Client will receive from the KDC the same group keying material it already has in memory.

Figure 14 gives an overview of the exchange described above, while Figure 15 shows an example.

```
Client                                                          KDC
   |                                                              |
   |------------------ Key Distribution Request: ---------------->|
   |           GET ace-group/GROUPNAME/nodes/NODENAME             |
   |                                                              |
   |<-------- Key Distribution Response: 2.05 (Content) ---------|
   |                                                              |
```

Figure 14: Message Flow of Key Distribution Request-Response

```
Request:

Header: GET (Code=0.01)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "nodes"
Uri-Path: "c101"
Payload: -

Response:

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation,
         with KEY and IND_KEY being CBOR byte strings,
         and "ind-key" the profile-specified label
         for individual keying material):
  { "gkty": 13, "key": KEY, "num": 12, "ind-key": IND_KEY }
```

Figure 15: Example of Key Distribution Request-Response

Alternatively, the re-distribution of keying material can be initiated by the KDC, which e.g.:

* Can make the ace-group/GROUPNAME resource Observable [RFC7641], and send notifications to Clients when the keying material is updated.

* Can send the payload of the Key Distribution Response in one or multiple multicast POST requests to the members of the group, using secure rekeying schemes such as [RFC2093][RFC2094][RFC2627].

* Can send unicast POST requests to each Client over a secure channel, with the same payload as the Key Distribution Response. When sending such requests, the KDC can target the URI path provided by the intended recipient upon joining the group, as specified in the 'control_path' parameter of the Joining Request (see Section 4.1.2.1).

* Can act as a publisher in a pub-sub scenario, and update the keying material by publishing on a specific topic on a broker, which all the members of the group are subscribed to.

Note that these methods of KDC-initiated key distribution have different security properties and require different security associations.

4.5. Requesting a Change of Keying Material

Beside possible expiration, the client may need to communicate to the KDC its need for the keying material to be renewed, e.g. due to exhaustion of AEAD nonces, if AEAD is used for protecting group communnication. Depending on the application profile (OPT8), this can result in renewal of individual keying material, group keying material, or both.

For example, if the Client uses an individual key to protect outgoing traffic and has to renew it, the node may request a new one, or new input material to derive it, without renewing the whole group keying material.

To this end, the client performs a Key Renewal Request/Response exchange with the KDC, i.e. it sends a CoAP PUT request to the /ace-group/GROUPNAME/nodes/NODENAME endpoint at the KDC, where GROUPNAME is the group name and NODENAME is its node name, and formatted as defined in Section 4.1.6.2.

Figure 16 gives an overview of the exchange described above, while Figure 17 shows an example.

```
Client                                                  KDC
   |                                                      |
   |------------------ Key Renewal Request: -------------->|
   |           PUT ace-group/GROUPNAME/nodes/NODENAME      |
   |                                                      |
   |<-------- Key Renewal Response: 2.05 (Content) --------|
   |                                                      |
```

Figure 16: Message Flow of Key Renewal Request-Response

```
     Request:

     Header: PUT (Code=0.03)
     Uri-Host: "kdc.example.com"
     Uri-Path: "ace-group"
     Uri-Path: "g1"
     Uri-Path: "nodes"
     Uri-Path: "c101"
     Payload: -

     Response:

     Header: Content (Code=2.05)
     Content-Format: "application/ace-groupcomm+cbor"
     Payload (in CBOR diagnostic notation, with IND_KEY being
              a CBOR byte string, and "ind-key" the profile-specified
              label for individual keying material):
       { "ind-key": IND_KEY }
```

Figure 17: Example of Key Renewal Request-Response

Note the difference between the Key Distribution Request and the Key Renewal Request: while the first one only triggers distribution (the renewal might have happened independently, e.g. because of expiration), the second one triggers the KDC to produce new individual keying material for the requesting node.

4.6. Retrieval of Public Keys and Roles for Group Members

In case the KDC maintains the public keys of group members, a node in the group can contact the KDC to request public keys and roles of either all group members or a specified subset, by sending a CoAP GET or FETCH request to the /ace-group/GROUPNAME/pub-key endpoint at the KDC, where GROUPNAME is the group name, and formatted as defined in Section 4.1.3.2 and Section 4.1.3.1.

Figure 18 and Figure 20 give an overview of the exchanges described above, while Figure 19 and Figure 21 show an example for each exchange.

```
      Client                                                     KDC
         |                                                        |
         |--Public Key Request: GET /ace-group/GROUPNAME/pub-key->|
         |                                                        |
         |<--------- Public Key Response: 2.05 (Content) ---------|
         |                                                        |
```

Figure 18: Message Flow of Public Key Exchange to Request All Members Public Keys

```
Request:

Header: GET (Code=0.01)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "pub-key"
Payload: -

Response:

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
  { "pub_keys": << [ PUB_KEY1, PUB_KEY2, PUB_KEY3 ] >>,
    "peer_roles": ["sender", ["sender", "receiver"], "receiver"] }
```

Figure 19: Example of Public Key Exchange to Request All Members Public Keys

```
Client                                                     KDC
   |                                                         |
   |-Public Key Request: FETCH /ace-group/GROUPNAME/pub-key->|
   |                                                         |
   |<--------- Public Key Response: 2.05 (Created) ----------|
   |                                                         |
```

Figure 20: Message Flow of Public Key Exchange to Request Specific Members Public Keys

```
Request:

Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "pub-key"
Content-Format: "application/ace-groupcomm+cbor"
Payload:
  { "get_pub_keys": [[], ["c3"]] }

Response:

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
  { "pub_keys": << [ PUB_KEY3 ] >>,
    "peer_roles": ["receiver"] }
```

Figure 21: Example of Public Key Exchange to Request Specific Members Public Keys

4.7. Update of Public Key

In case the KDC maintains the public keys of group members, a node in the group can contact the KDC to upload a new public key to use in the group, and replace the currently stored one.

To this end, the Client performs a Public Key Update Request/Response exchange with the KDC, i.e. it sends a CoAP POST request to the /ace-group/GROUPNAME/nodes/NODENAME/pub-key endpoint at the KDC, where GROUPNAME is the group name and NODENAME is its node name.

The request is formatted as specified in Section 4.1.7.1.

Figure Figure 22 gives an overview of the exchange described above, while Figure 23 shows an example.

```
Client                                                          KDC
 |                                                               |
 |-------------- Public Key Update Request: ---------------------->|
 |      POST ace-group/GROUPNAME/nodes/NODENAME/pub-key          |
 |                                                               |
 |<------- Public Key Update Response: 2.04 (Changed) -------------|
 |                                                               |
```

Figure 22: Message Flow of Public Key Update Request-Response

```
        Request:

        Header: POST (Code=0.02)
        Uri-Host: "kdc.example.com"
        Uri-Path: "ace-group"
        Uri-Path: "g1"
        Uri-Path: "nodes"
        Uri-Path: "c101"
        Uri-Path: "pub-key"
        Content-Format: "application/ace-groupcomm+cbor"
        Payload (in CBOR diagnostic notation, with PUB_KEY
                 and SIG being CBOR byte strings):
          { "client_cred": PUB_KEY, "cnonce": h'9ff7684414affcc8',
            "client_cred_verify": SIG }

        Response:

        Header: Changed (Code=2.04)
        Payload: -
```

Figure 23: Example of Public Key Update Request-Response

If the application requires backward security, the KDC MUST generate new group keying material and securely distribute it to all the current group members, upon a group member updating its own public key. To this end, the KDC uses the message format of the response defined in Section 4.1.2.2. Application profiles may define alternative ways of retrieving the keying material, such as sending separate requests to different resources at the KDC (Section 4.1.2.2, Section 4.1.3.2, Section 4.1.4.1). The KDC MUST increment the version number of the current keying material, before distributing the newly generated keying material to the group. After that, the KDC SHOULD store the distributed keying material in persistent storage.

Additionally, after updating its own public key, a group member MAY send a number of the later requests including an identifier of the updated public key, to signal nodes that they need to retrieve it. How that is done depends on the group communication protocol used, and therefore is application profile specific (OPT10).

4.8. Retrieval of Group Policies

A node in the group can contact the KDC to retrieve the current group policies, by sending a CoAP GET request to the /ace-group/GROUPNAME/policies endpoint at the KDC, where GROUPNAME is the group name, and formatted as defined in Section 4.1.4.1

Figure 24 gives an overview of the exchange described above, while Figure 25 shows an example.

```
    Client                                                  KDC
        |                                                    |
        |-Policies Request: GET ace-group/GROUPNAME/policies ->|
        |                                                    |
        |<--------- Policies Response: 2.05 (Content) ---------|
        |                                                    |
```

Figure 24: Message Flow of Policies Request-Response

```
Request:

Header: GET (Code=0.01)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "policies"
Payload: -

Response:

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload(in CBOR diagnostic notation):
  { "group_policies": {"exp-delta": 120} }
```

Figure 25: Example of Policies Request-Response

4.9. Retrieval of Keying Material Version

A node in the group can contact the KDC to request information about the version number of the symmetric group keying material, by sending a CoAP GET request to the /ace-group/GROUPNAME/num endpoint at the KDC, where GROUPNAME is the group name, formatted as defined in Section 4.1.5.1. In particular, the version is incremented by the KDC every time the group keying material is renewed, before it's distributed to the group members.

Figure 26 gives an overview of the exchange described above, while Figure 27 shows an example.

```
       Client                                                KDC
          |                                                    |
          |---- Version Request: GET ace-group/GROUPNAME/num ---->|
          |                                                    |
          |<--------- Version Response: 2.05 (Content) -----------|
          |                                                    |
```

Figure 26: Message Flow of Version Request-Response

```
                 Request:

                 Header: GET (Code=0.01)
                 Uri-Host: "kdc.example.com"
                 Uri-Path: "ace-group"
                 Uri-Path: "g1"
                 Uri-Path: "num"
                 Payload: -

                 Response:

                 Header: Content (Code=2.05)
                 Content-Format: text/plain
                 Payload(in CBOR diagnostic notation):
                   13
```

Figure 27: Example of Version Request-Response

4.10. Group Leaving Request

A node can actively request to leave the group. In this case, the Client sends a CoAP DELETE request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the KDC, where GROUPNAME is the group name and NODENAME is its node name, formatted as defined in Section 4.1.6.3

Alternatively, a node may be removed by the KDC, without having explicitly asked for it. This is further discussed in Section 5.

5. Removal of a Node from the Group

This section describes the different scenarios according to which a node ends up being removed from the group.

If the application requires forward security, the KDC MUST generate new group keying material and securely distribute it to all the current group members but the leaving node, using the message format of the Key Distribution Response (see Section 4.4). Application profiles may define alternative message formats. Before distributing the new group keying material, the KDC MUST increment the version number of the keying material.

Note that, after having left the group, a node may wish to join it again. Then, as long as the node is still authorized to join the group, i.e. it still has a valid access token, it can request to re-join the group directly to the KDC without needing to retrieve a new access token from the AS. This means that the KDC might decide to keep track of nodes with valid access tokens, before deleting all information about the leaving node.

A node may be evicted from the group in the following cases.

1. The node explicitly asks to leave the group, as defined in Section 4.10.

2. The node has been found compromised or is suspected so.

3. The node’s authorization to be a group member is not valid anymore, either because the access token has expired, or it has been revoked. If the AS provides Token introspection (see Section 5.7 of [I-D.ietf-ace-oauth-authz]), the KDC can optionally use it and check whether the node is still authorized for that group in that role.

In either case, once aware that a node is not authorized anymore, the KDC has to remove the unauthorized node from the list of group members, if the KDC keeps track of that.

In case of forced eviction, the KDC MAY explicitly inform the leaving node, if the Client implements the ’control_path’ resource specified in Section 4.1.2.1. To this end, the KDC MAY send a DEL request, targeting the URI specified in the ’control_path’ parameter of the Joining Request.

6. ACE Groupcomm Parameters

This specification defines a number of fields used during the second part of the message exchange, after the ACE Token POST exchange. The table below summarizes them, and specifies the CBOR key to use instead of the full descriptive name. Note that the media type ace-groupcomm+cbor MUST be used when these fields are transported.

Name	CBOR Key	CBOR Type	Reference
scope	TBD	byte string	Section 4.1.2.1
get_pub_keys	TBD	array / simple value null	Section 4.1.2.1, Section 4.1.3.1
client_cred	TBD	byte string	Section 4.1.2.1
cnonce	TBD	byte string	Section 4.1.2.1
client_cred_verify	TBD	byte string	Section 4.1.2.1
pub_keys_repos	TBD	text string	Section 4.1.2.1
control_path	TBD	text string	Section 4.1.2.1
gkty	TBD	integer / text string	Section 4.1.2.1
key	TBD	see "ACE Groupcomm Key" Registry	Section 4.1.2.1
num	TBD	int	Section 4.1.2.1
ace-groupcomm-profile	TBD	int	Section 4.1.2.1
exp	TBD	int	Section 4.1.2.1
pub_keys	TBD	byte string	Section 4.1.2.1
peer_roles	TBD	array	Section 4.1.2.1
group_policies	TBD	map	Section 4.1.2.1
mgt_key_material	TBD	byte string	Section 4.1.2.1
gid	TBD	array	Section 4.1.1.1
gname	TBD	array of text strings	Section 4.1.1.1
guri	TBD	array of text strings	Section 4.1.1.1

Table 1

7. Security Considerations

When a Client receives a message from a sender for the first time, it needs to have a mechanism in place to avoid replay, e.g. Appendix B.2 of [RFC8613]. In case the Client rebooted and lost the security state used to protect previous communication with that sender, such a mechanism is useful for the recipient to be on the safe side.

Besides, if the KDC has renewed the group keying material, and the time interval between the end of the rekeying process and the joining of the Client is sufficiently small, that Client is also on the safe side, since replayed older messages protected with the previous keying material will not be accepted.

The KDC must renew the group keying material upon its expiration.

The KDC should renew the keying material upon group membership change, and should provide it to the current group members through the rekeying scheme used in the group.

The KDC should renew the group keying material after rebooting, even in the case where all keying material is stored in persistent storage. However, if the KDC relies on Observe responses to notify the group of renewed keying material, after rebooting the KDC will have lost all the current ongoing Observations with the group members, and the previous keying material will be used to protect messages in the group anyway. The KDC will rely on each node requesting updates of the group keying material to establish the new keying material in the nodes, or, if implemented, it can push the update to the nodes in the group using the 'control_path' resource.

The KDC may enforce a rekeying policy that takes into account the overall time required to rekey the group, as well as the expected rate of changes in the group membership.

That is, the KDC may not rekey the group at every membership change, for instance if members' joining and leaving occur frequently and performing a group rekeying takes too long. The KDC may rekey the group after a minimum number of group members have joined or left within a given time interval, or after maximum amount of time since the last rekeying was completed, or yet during predictable network inactivity periods.

However, this would result in the KDC not constantly preserving backward and forward security. Newly joining group members could be able to access the keying material used before their joining, and thus could access past group communications. Also, until the KDC performs a group rekeying, the newly leaving nodes would still be able to access upcoming group communications that are protected with the keying material that has not yet been updated.

The KDC needs to have a mechanism in place to detect DoS attacks from nodes constantly initiating rekey events (for example by updating their public key), such as removing these nodes from the group.

The KDC also needs to have a congestion control mechanism in place to avoid network congestion when the KDC renews the group keying material; CoAP and Observe give guidance on such mechanisms, see Section 4.7 of [RFC7252] and Section 4.5.1 of [RFC7641].

7.1. Update of Keying Material

A group member can receive a message shortly after the group has been rekeyed, and new keying material has been distributed by the KDC. In the following two cases, this may result in misaligned keying material between the group members.

In the first case, the sender protects a message using the old keying material. However, the recipient receives the message after having received the new keying material, hence not being able to correctly process it. A possible way to ameliorate this issue is to preserve the old, recent, keying material for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained keying material. Note that a former (compromised) group member can take advantage of this by sending messages protected with the old retained keying material. Therefore, a conservative application policy should not admit the storage of old keying material.

In the second case, the sender protects a message using the new keying material, but the recipient receives that request before having received the new keying material. Therefore, the recipient would not be able to correctly process the request and hence discards it. If the recipient receives the new keying material shortly after that and the application at the sender endpoint performs retransmissions, the former will still be able to receive and correctly process the message. In any case, the recipient should actively ask the KDC for an updated keying material according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

A node that has left the group should not expect any of its outgoing messages to be successfully processed, if received after its leaving, due to a possible group rekeying occurred before the message reception.

7.2. Block-Wise Considerations

If the block-wise options [RFC7959] are used, and the keying material is updated in the middle of a block-wise transfer, the sender of the blocks just changes the keying material to the updated one and continues the transfer. As long as both sides get the new keying material, updating the keying material in the middle of a transfer will not cause any issue. Otherwise, the sender will have to transmit the message again, when receiving an error message from the recipient.

Compared to a scenario where the transfer does not use block-wise, depending on how fast the keying material is changed, the nodes might consume a larger amount of the network bandwidth resending the blocks again and again, which might be problematic.

8. IANA Considerations

This document has the following actions for IANA.

8.1. Media Type Registrations

This specification registers the 'application/ace-groupcomm+cbor' media type for messages of the protocols defined in this document following the ACE exchange and carrying parameters encoded in CBOR. This registration follows the procedures specified in [RFC6838].

Type name: application

Subtype name: ace-groupcomm+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as CBOR map containing the protocol parameters defined in [this document].

Security considerations: See Section 7 of this document.

Interoperability considerations: n/a

Published specification: [this document]

Applications that use this media type: The type is used by authorization servers, clients and resource servers that support the ACE groupcomm framework as specified in [this document].

Additional information: n/a

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

Intended usage: COMMON

Restrictions on usage: None

Author: Francesca Palombini francesca.palombini@ericsson.com (mailto:francesca.palombini@ericsson.com)

Change controller: IESG

8.2. CoAP Content-Formats Registry

This specification registers the following entry to the "CoAP Content-Formats" registry, within the "CoRE Parameters" registry:

Media Type: application/ace-groupcomm+cbor

Encoding: -

ID: TBD

Reference: [this document]

8.3. OAuth Parameters Registry

The following registrations are done for the OAuth ParametersRegistry following the procedure specified in section 11.2 of [RFC6749]:

o Parameter name: sign_info o Parameter usage location: token request, token response o Change Controller: IESG o Specification Document(s): [[This specification]]

o Parameter name: kdcchallenge o Parameter usage location: token response o Change Controller: IESG o Specification Document(s): [[This specification]]

8.4. OAuth Parameters CBOR Mappings Registry

The following registrations are done for the OAuth Parameters CBOR Mappings Registry following the procedure specified in section 8.9 of [I-D.ietf-ace-oauth-authz]:

* Name: sign_info
* CBOR Key: TBD (range -256 to 255)
* Value Type: any
* Reference: \[\[This specification\]\]

* Name: kdcchallenge
* CBOR Key: TBD (range -256 to 255)
* Value Type: byte string
* Reference: \[\[This specification\]\]

8.5. ACE Groupcomm Parameters Registry

This specification establishes the "ACE Groupcomm Parameters" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 8.11.

The columns of this Registry are:

* Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.

* CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.

* CBOR Type: This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.

* Reference: This contains a pointer to the public specification for the item.

This Registry has been initially populated by the values in Section 6. The Reference column for all of these entries refers to sections of this document.

8.6. ACE Groupcomm Key Registry

This specification establishes the "ACE Groupcomm Key" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 8.11.

The columns of this Registry are:

* Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.

* Key Type Value: This is the value used to identify the keying material. These values MUST be unique. The value can be a positive integer, a negative integer, or a text string.

* Profile: This field may contain one or more descriptive strings of application profiles to be used with this item. The values should be taken from the Name column of the "ACE Groupcomm Profile" Registry.

* Description: This field contains a brief description of the keying material.

* References: This contains a pointer to the public specification for the format of the keying material, if one exists.

This Registry has been initially populated by the values in Figure 8. The specification column for all of these entries will be this document.

8.7. ACE Groupcomm Profile Registry

This specification establishes the "ACE Groupcomm Profile" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 8.11. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

* Name: The name of the application profile, to be used as value of the profile attribute.

* Description: Text giving an overview of the application profile and the context it is developed for.

* CBOR Value: CBOR abbreviation for the name of this application profile. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

* Reference: This contains a pointer to the public specification of the abbreviation for this application profile, if one exists.

8.8. ACE Groupcomm Policy Registry

This specification establishes the "ACE Groupcomm Policy" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 8.11. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

* Name: The name of the group communication policy.

* CBOR label: The value to be used to identify this group communication policy. Key map labels MUST be unique. The label can be a positive integer, a negative integer or a string. Integer values between 0 and 255 and strings of length 1 are designated as Standards Track Document required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values of 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: the CBOR type used to encode the value of this group communication policy.

* Description: This field contains a brief description for this group communication policy.

* Reference: This field contains a pointer to the public specification providing the format of the group communication policy, if one exists.

This registry will be initially populated by the values in Figure 9.

8.9. Sequence Number Synchronization Method Registry

This specification establishes the "Sequence Number Synchronization Method" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 8.11. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

* Name: The name of the sequence number synchronization method.

* Value: The value to be used to identify this sequence number synchronization method.

* Description: This field contains a brief description for this sequence number synchronization method.

* Reference: This field contains a pointer to the public specification describing the sequence number synchronization method.

8.10. Interface Description (if=) Link Target Attribute Values Registry

This specification registers the following entry to the "Interface Description (if=) Link Target Attribute Values Registry" registry, within the "CoRE Parameters" registry:

* Attribute Value: ace.group

* Description: The 'ace group' interface is used to provision keying material and related informations and policies to members of a group using the Ace framework.

* Reference: [This Document]

8.11. Expert Review Instructions

The IANA Registries established in this document are defined as expert review. 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.

9. References

9.1. 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-35, 24 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-35.txt>.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", Work in Progress, Internet-Draft,

draft-ietf-cbor-7049bis-16, 30 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cbor-7049bis-16.txt>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-09, 23 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-oscore-groupcomm-09.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, <http://www.ietf.org/internet-drafts/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-14, 24 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cose-rfc8152bis-struct-14.txt>.

[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>.

[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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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>.

9.2. Informative References

[I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", Work in Progress, Internet-Draft, draft-ietf-ace-aif-00, 29 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-aif-00.txt>.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-14, 29 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-14.txt>.

[I-D.ietf-ace-mqtt-tls-profile]
Sengul, C. and A. Kirby, "Message Queuing Telemetry Transport (MQTT)-TLS profile of Authentication and Authorization for Constrained Environments (ACE) Framework", Work in Progress, Internet-Draft, draft-ietf-ace-mqtt-tls-profile-08, 1 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-mqtt-tls-profile-08.txt>.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-13, 27 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-13.txt>.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <http://www.ietf.org/internet-drafts/draft-ietf-core-coap-pubsub-09.txt>.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-01, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-groupcomm-bis-01.txt>.

[RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <https://www.rfc-editor.org/info/rfc2093>.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <https://www.rfc-editor.org/info/rfc2094>.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <https://www.rfc-editor.org/info/rfc2627>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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>.

Appendix A. Requirements on Application Profiles

This section lists the requirements on application profiles of this specification,for the convenience of application profile designers.

* REQ1: If the value of the GROUPNAME URI path and the group name in the access token scope (gname in Section 3.2) don't match, specify the mechanism to map the GROUPNAME value in the URI to the group name (REQ1) (see Section 4.1).

* REQ2: Specify the encoding and value of roles, for scope entries of 'scope' (see Section 3.1).

* REQ3: If used, specify the acceptable values for 'sign_alg' (see Section 3.3).

* REQ4: If used, specify the acceptable values for 'sign_parameters' (see Section 3.3).

* REQ5: If used, specify the acceptable values for 'sign_key_parameters' (see Section 3.3).

* REQ6: If used, specify the acceptable values for 'pub_key_enc' (see Section 3.3).

* REQ7a: Register a Resource Type for the root url-path, which is used to discover the correct url to access at the KDC (see Section 4.1).

* REQ7aa: Define what operations (i.e. CoAP methods) are allowed on each resource, for each role defined in REQ2 (see Section 3.3).

* REQ7b: Specify the exact encoding of group identifier (see Section 4.1.1.1).

* REQ7: Specify the exact format of the 'key' value (see Section 4.1.2.1).

* REQ8: Specify the acceptable values of 'gkty' (see Section 4.1.2.1).

* REQ9: Specify the format of the identifiers of group members (see Section 4.1.2.1).

* REQ10: Specify the communication protocol the members of the group must use (e.g., multicast CoAP).

* REQ11: Specify the security protocol the group members must use to protect their communication (e.g., group OSCORE). This must provide encryption, integrity and replay protection.

* REQ12: Specify and register the application profile identifier (see Section 4.1.2.1).

* REQ13: Specify policies at the KDC to handle ids that are not included in get_pub_keys (see Section 4.1.3.1).

* REQ14: If used, specify the format and content of 'group_policies' and its entries. Specify the policies default values (see Section 4.1.2.1).

* REQ15: Specify the format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label (see Section 4.1.6.2).

* REQ16: Specify how the communication is secured between Client and KDC. Optionally, specify tranport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and KDC (see Section 4.3.

* REQ17: Specify how the nonce N_S is generated, if the token was not posted (e.g. if it is used directly to validate TLS instead).

* REQ18: Specify if 'mgt_key_material' used, and if yes specify its format and content (see Section 4.1.2.1). If the usage of 'mgt_key_material' is indicated and its format defined for a specific key management scheme, that format must explicitly indicate the key management scheme itself. If a new rekeying scheme is defined to be used for an existing 'mgt_key_material' in an existing profile, then that profile will have to be updated accordingly, especially with respect to the usage of 'mgt_key_material' related format and content.

* REQ19: Define the initial value of the 'num' parameter (sse Section 4.1.2.1).

* OPT1: Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used (see Section 4.1.2.1).

* OPT2a: Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' is not used (see Section 3.3).

* OPT2b: Optionally, specify the additional parameters used in the Token Post exchange (see Section 3.3).

* OPT3: Optionally, specify the encoding of 'pub_keys_repos' if the default is not used (see Section 4.1.2.1).

* OPT4: Optionally, specify policies that instruct clients to retain messages and for how long, if they are unsuccessfully decrypted (see Section 4.4). This makes it possible to decrypt such messages after getting updated keying material.

* OPT5: Optionally, specify the behavior of the handler in case of failure to retrieve a public key for the specific node (see Section 4.1.2.1).

* OPT6: Optionally, specify possible or required payload formats for specific error cases.

* OPT7: Optionally, specify CBOR values to use for abbreviating identifiers of roles in the group or topic (see Section 3.1).

* OPT8: Optionally, specify for the KDC to perform group rekeying (together or instead of renewing individual keying material) when receiving a Key Renewal Request (see Section 4.5).

* OPT9: Optionally, specify the functionalities implemented at the 'control_path' resource hosted at the Client, including message exchange encoding and other details (see Section 4.1.2.1).

* OPT10: Optionally, specify how the identifier of the sender's public key is included in the group request (see Section 4.7).

Appendix B. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

B.1. Version -04 to -05

* Updated uppercase/lowercase URI segments for KDC resources.

* Supporting single Access Token for multiple groups/topics.

* Added 'control_path' parameter in the Joining Request.

* Added 'peer_roles' parameter to support legal requesters/ responders.

* Clarification on stopping using owned keying material.

* Clarification on different reasons for processing failures, related policies, and requirement OPT4.

* Added a KDC sub-resource for group members to upload a new public key.

* Possible group rekeying following an individual Key Renewal Request.

* Clarified meaning of requirement REQ3; added requirement OPT8.

* Editorial improvements.

B.2. Version -03 to -04

* Revised RESTful interface, as to methods and parameters.

* Extended processing of joining request, as to check/retrieval of public keys.

* Revised and extended profile requirements.

* Clarified specific usage of parameters related to signature algorithms/keys.

* Included general content previously in draft-ietf-ace-key-groupcomm-oscore

* Registration of media type and content format application/ace-group+cbor

* Editorial improvements.

B.3. Version -02 to -03

* Exchange of information on the countersignature algorithm and related parameters, during the Token POST (Section 3.3).

* Restructured KDC interface, with new possible operations (Section 4).

* Client PoP signature for the Joining Request upon joining (Section 4.1.2.1).

* Revised text on group member removal (Section 5).

* Added more profile requirements (Appendix A).

B.4. Version -01 to -02

* Editorial fixes.

* Distinction between transport profile and application profile (Section 1.1).

* New parameters 'sign_info' and 'pub_key_enc' to negotiate parameter values for signature algorithm and signature keys (Section 3.3).

* New parameter 'type' to distinguish different Key Distribution Request messages (Section 4.1).

* New parameter 'client_cred_verify' in the Key Distribution Request to convey a Client signature (Section 4.1).

* Encoding of 'pub_keys_repos' (Section 4.1).

* Encoding of 'mgt_key_material' (Section 4.1).

* Improved description on retrieval of new or updated keying material (Section 6).

* Encoding of 'get_pub_keys' in Public Key Request (Section 7.1).

* Extended security considerations (Sections 10.1 and 10.2).

* New "ACE Public Key Encoding" IANA Registry (Section 11.2).

* New "ACE Groupcomm Parameters" IANA Registry (Section 11.3), populated with the entries in Section 8.

* New "Ace Groupcomm Request Type" IANA Registry (Section 11.4), populated with the values in Section 9.

* New "ACE Groupcomm Policy" IANA Registry (Section 11.7) populated with two entries "Sequence Number Synchronization Method" and "Key Update Check Interval" (Section 4.2).

* Improved list of requirements for application profiles (Appendix A).

B.5. Version -00 to -01

* Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).

* Defined error handling on the KDC (Sections 4.2 and 6.2).

* Updated format of the Key Distribution Response as a whole (Section 4.2).

* Generalized format of 'pub_keys' in the Key Distribution Response (Section 4.2).

* Defined format for the message to request leaving the group (Section 5.2).

* Renewal of individual keying material and methods for group rekeying initiated by the KDC (Section 6).

* CBOR type for node identifiers in 'get_pub_keys' (Section 7.1).

* Added section on parameter identifiers and their CBOR keys (Section 8).

* Added request types for requests to a Join Response (Section 9).

* Extended security considerations (Section 10).

* New IANA registries "ACE Groupcomm Key Registry", "ACE Groupcomm Profile Registry", "ACE Groupcomm Policy Registry" and "Sequence Number Synchronization Method Registry" (Section 11).

* Added appendix about requirements for application profiles of ACE on group communication (Appendix A).

Acknowledgments

The following individuals were helpful in shaping this document: Christian Amsuess, Carsten Bormann, Rikard Hoeglund, Ben Kaduk, John Mattsson, Daniel Migault, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; by the H2020 project SIFIS-Home (Grant agreement 952652); and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden

Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
J. Park
Universitaet Duisburg-Essen
F. Palombini
Ericsson AB

November 02, 2020

Key Management for OSCORE Groups in ACE
draft-ietf-ace-key-groupcomm-oscore-09

Abstract

This specification defines an application profile of the ACE framework for Authentication and Authorization, to request and provision keying material in group communication scenarios that are based on CoAP and secured with Group Object Security for Constrained RESTful Environments (OSCORE). This application profile delegates the authentication and authorization of Clients that join an OSCORE group through a Resource Server acting as Group Manager for that group. This application profile leverages protocol-specific transport profiles of ACE to achieve communication security, server authentication and proof-of-possession for a key owned by the Client and bound to an OAuth 2.0 Access Token.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   3
     1.1.  Terminology . . . . . . . . . . . . . . . . . . . . . . .   4
   2.  Protocol Overview . . . . . . . . . . . . . . . . . . . . . .   5
     2.1.  Overview of the Joining Process . . . . . . . . . . . . .   6
     2.2.  Overview of the Group Rekeying Process  . . . . . . . . .   6
   3.  Format of Scope . . . . . . . . . . . . . . . . . . . . . . .   7
   4.  Joining Node to Authorization Server  . . . . . . . . . . . .   8
     4.1.  Authorization Request . . . . . . . . . . . . . . . . . .   9
     4.2.  Authorization Response  . . . . . . . . . . . . . . . . .   9
   5.  Interface at the Group Manager  . . . . . . . . . . . . . . .   9
     5.1.  ace-group/GROUPNAME/active  . . . . . . . . . . . . . . .  10
       5.1.1.  GET Handler . . . . . . . . . . . . . . . . . . . . .  10
     5.2.  Admitted Methods  . . . . . . . . . . . . . . . . . . . .  11
   6.  Token POST and Group Joining  . . . . . . . . . . . . . . . .  11
     6.1.  Token Post  . . . . . . . . . . . . . . . . . . . . . . .  12
       6.1.1.  'ecdh_info' Parameter . . . . . . . . . . . . . . . .  14
     6.2.  Sending the Joining Request . . . . . . . . . . . . . . .  16
       6.2.1.  Value of the N_S Challenge  . . . . . . . . . . . . .  17
     6.3.  Processing the Joining Request  . . . . . . . . . . . . .  17
     6.4.  Joining Response  . . . . . . . . . . . . . . . . . . . .  19
   7.  Public Keys of Joining Nodes  . . . . . . . . . . . . . . . .  23
   8.  Retrieval of Updated Keying Material  . . . . . . . . . . . .  25
     8.1.  Retrieval of Group Keying Material  . . . . . . . . . . .  25
     8.2.  Retrieval of Group Keying Material and Sender ID  . . . .  26
   9.  Requesting a Change of Keying Material  . . . . . . . . . . .  26
   10. Retrieval of Public Keys and Roles for Group Members  . . . .  27
   11. Update of Public Key  . . . . . . . . . . . . . . . . . . . .  28
   12. Retrieval of Group Policies . . . . . . . . . . . . . . . . .  29
   13. Retrieval of Keying Material Version  . . . . . . . . . . . .  29
   14. Retrieval of Group Status . . . . . . . . . . . . . . . . . .  29
   15. Retrieval of Group Names and URIs . . . . . . . . . . . . . .  30
   16. Request to Leave the Group  . . . . . . . . . . . . . . . . .  32
   17. Removal of a Group Member . . . . . . . . . . . . . . . . . .  32
   18. Group Rekeying Process  . . . . . . . . . . . . . . . . . . .  33
   19. Default Values for Group Configuration Parameters . . . . . .  35
   20. Security Considerations . . . . . . . . . . . . . . . . . . .  38
```

20.1. Management of OSCORE Groups 38
20.2. Size of Nonces for Signature Challenge 39
20.3. Reusage of Nonces for Signature Challenge 40
21. IANA Considerations 40
21.1. ACE Groupcomm Parameters Registry 40
21.2. ACE Groupcomm Key Registry 41
21.3. ACE Groupcomm Profile Registry 41
21.4. Sequence Number Synchronization Method Registry 41
21.5. OSCORE Security Context Parameters Registry 42
21.6. TLS Exporter Label Registry 44
21.7. AIF Registry 45
21.8. Media Type Registrations 45
21.9. CoAP Content-Format Registry 46
21.10. Group OSCORE Roles Registry 46
21.11. CoRE Resource Type Registry 47
21.12. Expert Review Instructions 47
22. References 48
22.1. Normative References 48
22.2. Informative References 51
22.3. URIs 52
Appendix A. Profile Requirements 52
Appendix B. Document Updates 55
B.1. Version -08 to -09 56
B.2. Version -07 to -08 56
B.3. Version -06 to -07 57
B.4. Version -05 to -06 57
B.5. Version -04 to -05 58
B.6. Version -03 to -04 58
B.7. Version -02 to -03 59
B.8. Version -01 to -02 59
B.9. Version -00 to -01 60
Acknowledgments 60
Authors' Addresses 61

1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] is a method for application-layer protection of the Constrained Application Protocol (CoAP) [RFC7252], using CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and enabling end-to-end security of CoAP payload and options.

As described in [I-D.ietf-core-oscore-groupcomm], Group OSCORE is used to protect CoAP group communication over IP multicast [I-D.ietf-core-groupcomm-bis]. This relies on a Group Manager, which is responsible for managing an OSCORE group and enables the group members to exchange CoAP messages secured with Group OSCORE. The

Group Manager can be responsible for multiple groups, coordinates the joining process of new group members, and is entrusted with the distribution and renewal of group keying material.

This specification is an application profile of [I-D.ietf-ace-key-groupcomm], which itself builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz]. Message exchanges among the participants as well as message formats and processing follow what specified in [I-D.ietf-ace-key-groupcomm] for provisioning and renewing keying material in group communication scenarios, where Group OSCORE is used to protect CoAP group communication over IP multicast.

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:

- o The terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] and in the Authorization Information Format (AIF) [I-D.ietf-ace-aif] to express authorization information. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

- o The terms and concept related to the message formats and processing specified in [I-D.ietf-ace-key-groupcomm], for provisioning and renewing keying material in group communication scenarios.

- o The terms and concepts described in CBOR [I-D.ietf-cbor-7049bis] and COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].

- o The terms and concepts decribed in CoAP [RFC7252] and group communication for CoAP [I-D.ietf-core-groupcomm-bis]. Unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

- The terms and concepts for protection and processing of CoAP messages through OSCORE [RFC8613] and through Group OSCORE [I-D.ietf-core-oscore-groupcomm] in group communication scenarios. These include the concept of Group Manager, as the entity responsible for a set of groups where communications are secured with Group OSCORE. In this specification, the Group Manager acts as Resource Server.

Additionally, this document makes use of the following terminology.

- Requester: member of an OSCORE group that sends request messages to other members of the group.

- Responder: member of an OSCORE group that receives request messages from other members of the group. A responder may reply back, by sending a response message to the requester which has sent the request message.

- Monitor: member of an OSCORE group that is configured as responder and never replies back to requesters after receiving request messages. This corresponds to the term "silent server" used in [I-D.ietf-core-oscore-groupcomm].

- Signature verifier: entity external to the OSCORE group and intended to verify the countersignature of messages exchanged in the group. An authorized signature verifier does not join the OSCORE group as an actual member, yet it can retrieve the public keys of the current group members from the Group Manager.

2. Protocol Overview

Group communication for CoAP over IP multicast has been enabled in [I-D.ietf-core-groupcomm-bis] and can be secured with Group Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] as described in [I-D.ietf-core-oscore-groupcomm]. A network node joins an OSCORE group by interacting with the responsible Group Manager. Once registered in the group, the new node can securely exchange messages with other group members.

This specification describes how to use [I-D.ietf-ace-key-groupcomm] and [I-D.ietf-ace-oauth-authz] to perform a number of authentication, authorization and key distribution actions, as defined in Section 2 of [I-D.ietf-ace-key-groupcomm], for an OSCORE group.

With reference to [I-D.ietf-ace-key-groupcomm]:

- The node wishing to join the OSCORE group, i.e. the joining node, is the Client.

- o The Group Manager is the Key Distribution Center (KDC), acting as a Resource Server.

- o The Authorization Server associated to the Group Manager is the AS.

All communications between the involved entities MUST be secured.

In particular, communications between the Client and the Group Manager leverage protocol-specific transport profiles of ACE to achieve communication security, proof-of-possession and server authentication. It is expected that, in the commonly referred base-case of this specification, the transport profile to use is pre-configured and well-known to nodes participating in constrained applications.

Appendix A lists the specifications on this application profile of ACE, based on the requirements defined in Appendix A of [I-D.ietf-ace-key-groupcomm].

2.1. Overview of the Joining Process

A node performs the steps described in Section 4.3 of [I-D.ietf-ace-key-groupcomm] in order to join an OSCORE group. The format and processing of messages exchanged among the participants are further specified in Section 4 and Section 6 of this document.

2.2. Overview of the Group Rekeying Process

If the application requires backward and forward security, the Group Manager MUST generate new keying material and distribute it to the group (rekeying) upon membership changes.

That is, the group is rekeyed when a node joins the group as a new member, or after a current member leaves the group. By doing so, a joining node cannot access communications in the group prior its joining, while a leaving node cannot access communications in the group after its leaving.

The keying material distributed through a group rekeying MUST include:

- o A new Group Identifier (Gid) for the group as introduced in [I-D.ietf-ace-key-groupcomm], used as ID Context parameter of the Group OSCORE Common Security Context of that group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

Note that the Gid differs from the group name also introduced in [I-D.ietf-ace-key-groupcomm], which is a plain, stable and invariant identifier, with no cryptographic relevance and meaning.

- o A new value for the Master Secret parameter of the Group OSCORE Common Security Context of that group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

Also, the distributed keying material MAY include a new value for the Master Salt parameter of the Group OSCORE Common Security Context of that group.

Upon generating the new group keying material and before starting its distribution, the Group Manager MUST increment the version number of the group keying material. When rekeying a group, the Group Manager MUST preserve the current value of the Sender ID of each member in that group.

The Group Manager MUST support the Group Rekeying Process described in Section 18. Future application profiles may define alternative message formats and distribution schemes to perform group rekeying.

3. Format of Scope

Building on Section 3.1 of [I-D.ietf-ace-key-groupcomm], this section defines the exact format and encoding of scope to use.

To this end, this profile uses the Authorization Information Format (AIF) [I-D.ietf-ace-aif], and defines the following AIF specific data model AIF-OSCORE-GROUPCOMM.

With reference to the generic AIF model

```
AIF-Generic<Toid, Tperm> = [* [Toid, Tperm]]
```

the value of the CBOR byte string used as scope encodes the CBOR array [* [Toid, Tperm]], where each [Toid, Tperm] element corresponds to one scope entry.

Then, for each scope entry:

- o the object identifier ("Toid") is specialized as a CBOR text string, specifying the group name for the scope entry;
- o the permission set ("Tperm") is specialized as a CBOR unsigned integer with value R, specifying the role(s) that the client wishes to take in the group (REQ2). The value R is computed as follows:

* each role in the permission set is converted into the corresponding numeric identifier X from the "Value" column of the table in Figure 1.

* the set of N numbers is converted into the single value R, by taking each numeric identifier X_1, X_2, ..., X_N to the power of two, and then computing the inclusive OR of the binary representations of all the power values.

Name	Value	Description
Reserved	0	This value is reserved
Requester	1	Send requests; receive responses
Responder	2	Send responses; receive requests
Monitor	3	Receive requests; never send requests/responses
Verifier	4	Verify countersignature of intercepted messages

Figure 1: Numeric identifier of roles in the OSCORE group

The CDDL [RFC8610] definition of the AIF-OSCORE-GROUPCOMM data model is as follows:

```
AIF-OSCORE-GROUPCOMM = AIF_Generic<path, permissions>

path = tstr  ; Group name
permissions = uint . bits roles
roles = &(
   Requester: 1,
   Responder: 2,
   Monitor: 3,
   Verifier: 4
)
```

Future specifications that define new roles MUST register a corresponding numeric identifier in the "Group OSCORE Roles" Registry defined in Section 21.10 of this specification.

4. Joining Node to Authorization Server

This section describes how the joining node interacts with the AS in order to be authorized to join an OSCORE group under a given Group

Manager. In particular, it considers a joining node that intends to contact that Group Manager for the first time.

The message exchange between the joining node and the AS consists of the messages Authorization Request and Authorization Response defined in Section 3 of [I-D.ietf-ace-key-groupcomm]. Note that what is defined in [I-D.ietf-ace-key-groupcomm] applies, and only additions or modifications to that specification are defined here.

4.1. Authorization Request

The Authorization Request message is as defined in Section 3.1 of [I-D.ietf-ace-key-groupcomm], with the following additions.

- If the 'scope' parameter is present:

  * The value of the CBOR byte string encodes a CBOR array, whose format MUST follow the data model AIF-OSCORE-GROUPCOMM defined in Section 3. In particular, for each OSCORE group to join:

    + The group name is encoded as a CBOR text string.

    + The set of requested roles is expressed as a single CBOR unsigned integer, computed as defined in Section 3 (REQ2) from the numerical abbreviations defined in Figure 1 for each requested role (OPT7).

4.2. Authorization Response

The Authorization Response message is as defined in Section 3.2 of [I-D.ietf-ace-key-groupcomm], with the following additions:

- The AS MUST include the 'expires_in' parameter. Other means for the AS to specify the lifetime of Access Tokens are out of the scope of this specification.

- The AS MUST include the 'scope' parameter, when the value included in the Access Token differs from the one specified by the joining node in the request. In such a case, the second element of each scope entry MUST be present, and specifies the set of roles that the joining node is actually authorized to take in the OSCORE group for that scope entry, encoded as specified in Section 4.1.

5. Interface at the Group Manager

The Group Manager provides the interface defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm], with one additional sub-resource defined in Section 5.1 of this specification.

Furthermore, Section 5.2 provides a summary of the CoAP methods admitted to access different resources at the Group Manager, for nodes with different roles in the group or as non members (REQ7aa).

The GROUPNAME segment of the URI path MUST match with the group name specified in the scope entry of the Access Token scope (i.e. 'gname' in Section 3.1 of [I-D.ietf-ace-key-groupcomm]) (REQ1).

The Resource Type (rt=) Link Target Attribute value "core.osc.gm" is registered in Section 21.11 (REQ7a), and can be used to describe group-membership resources and its sub-resources at a Group Manager, e.g. by using a link-format document [RFC6690].

Applications can use this common resource type to discover links to group-membership resources for joining OSCORE groups, e.g. by using the approach described in [I-D.tiloca-core-oscore-discovery].

5.1. ace-group/GROUPNAME/active

This resource implements a GET handler.

5.1.1. GET Handler

The handler expects a GET request.

The Group Manager verifies that the group name in the /ace-group/GROUPNAME/active path is a subset of the 'scope' stored in the Access Token associated to the requesting client.

The Group Manager also verifies that the roles granted to the requesting client in the group allow it to perform this operation on this resource (REQ7aa). If either verification fails, the Group Manager MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the requesting client is a current member of the group. If verification fails, the Group Manager MUST respond with a 4.01 (Unauthorized) error message.

If verification succeeds, the handler returns a 2.05 (Content) message containing the CBOR simple value True if the group is currently active, or the CBOR simple value False otherwise. The group is considered active if it is set to allow new members to join, and if communication within the group is fine to happen.

The method to set the current group status, i.e. active or inactive, is out of the scope of this specification, and is defined for the administrator interface of the Group Manager specified in [I-D.ietf-ace-oscore-gm-admin].

5.2. Admitted Methods

The table in Figure 2 summarizes the CoAP methods admitted to access different resources at the Group Manager, for (non-)members of a group with group name GROUPNAME, and considering different roles. The last two rows of the table apply to a node with node name NODENAME.

Resource	Type1	Type2	Type3	Type4
ace-group/	F	F	F	-
ace-group/GROUPNAME/	G Po	G Po	Po *	Po
ace-group/GROUPNAME/active	G	G	-	-
ace-group/GROUPNAME/pub-key	G F	G F	G F	-
ace-group/GROUPNAME/policies	G	G	-	-
ace-group/GROUPNAME/num	G	G	-	-
ace-group/GROUPNAME/nodes/ NODENAME	G Pu D	G D	-	-
ace-group/GROUPNAME/nodes/ NODENAME/pub-key	Po	-	-	-

Type1 = Member as Requester and/or Responder	G = GET
Type2 = Member as Monitor	F = FETCH
Type3 = Non-member / Authorized to be Verifier	Po = POST
(*) = cannot join the group as Verifier	Pu = PUT
Type4 = Non-member / Not authorized to be Verifier	D = DELETE

Figure 2: Admitted CoAP Methods on the Group Manager Resources

6. Token POST and Group Joining

The rest of this section describes the interactions between the joining node and the Group Manager, i.e. the sending of the Access Token and the Request-Response exchange to join the OSCORE group. The message exchange between the joining node and the Group Manager consists of the messages defined in Section 3.3 and 4.3 of [I-D.ietf-ace-key-groupcomm]. Note that what is defined in [I-D.ietf-ace-key-groupcomm] applies, and only additions or modifications to that specification are defined here.

A signature verifier provides the Group Manager with an Access Token, as described in Section 6.1, just as any another joining node does. However, unlike candidate group members, it does not join any OSCORE group, i.e. it does not perform the joining process defined in Section 6.2. After successfully posting an Access Token, a signature verifier is authorized to perform only the operations specified in Section 10, to retrieve the public keys of group members, and only for the OSCORE groups specified in the validated Access Token. The Group Manager MUST respond with a 4.01 (Unauthorized) error message, in case a signature verifier attempts to access any other endpoint than /ace-group/GROUPNAME/pub-key at the Group Manager.

6.1. Token Post

The Token post exchange is defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm].

Additionally to what defined in [I-D.ietf-ace-key-groupcomm], the following applies.

o The CoAP POST request MAY additionally contain the following parameter, which, if included, MUST have the corresponding values:

  * 'ecdh_info' defined in Section 6.1.1, encoding the CBOR simple value Null to require information on the ECDH algorithm, the ECDH algorithm parameters, the ECDH key parameters and on the exact encoding of public keys used in the group, in case the joining node supports the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

  Alternatively, the joining node may retrieve this information by other means.

o The 'kdcchallenge' parameter contains a dedicated nonce N_S generated by the Group Manager. For the N_S value, it is RECOMMENDED to use a 8-byte long random nonce. The joining node can use this nonce in order to prove the possession of its own private key, upon joining the group (see Section 6.2).

  The 'kdcchallenge' parameter MAY be omitted from the 2.01 (Created) response, if the 'scope' of the Access Token specifies only the role "monitor" or only the role "verifier" or both of them, for each and every of the specified groups.

o If the 'sign_info' parameter is present in the response, the following applies for each element 'sign_info_entry'.

   * 'sign_alg' takes value from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms].

   * 'sign_parameters' is a CBOR array including the following two elements:

      + 'sign_alg_capab', encoded as a CBOR array. Its format and value are the same of the COSE capabilities for the algorithm indicated in 'sign_alg', as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms] (REQ4).

      + 'sign_key_type_capab', encoded as a CBOR array. Its format and value are the same of the COSE capabilities for the COSE key type of the keys used with the algorithm indicated in 'sign_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (REQ4).

   * 'sign_key_parameters' is a CBOR array. Its format and value are the same of the COSE capabilities for the COSE key type of the keys used with the algorithm indicated in 'sign_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types] (REQ5).

   * 'pub_key_enc' takes value 1 ("COSE_Key") from the 'Confirmation Key' column of the "CWT Confirmation Method" Registry [CWT.Confirmation.Methods], so indicating that public keys in the OSCORE group are encoded as COSE Keys [I-D.ietf-cose-rfc8152bis-struct]. Future specifications may define additional values for this parameter.

o If 'ecdh_info' is included in the request, the Group Manager MAY include the 'ecdh_info' parameter defined in Section 6.1.1, with the same encoding. Note that the field 'id' takes as value the group name, or array of group names, for which the corresponding 'ecdh_info_entry' applies to.

Note that, other than through the above parameters as defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm], the joining node MAY have previously retrieved this information by other means, e.g. by using the approach described in [I-D.tiloca-core-oscore-discovery] to discover the OSCORE group and the link to the associated group-membership resource at the Group Manager (OPT2a).

Additionally, if allowed by the used transport profile of ACE, the joining node may instead provide the Access Token to the Group

Manager by other means, e.g. during a secure session establishment (see Section 3.3.2 of [I-D.ietf-ace-dtls-authorize]).

6.1.1. 'ecdh_info' Parameter

The 'ecdh_info' parameter is an OPTIONAL parameter of the Token Post response message defined in Section 5.1.2. of [I-D.ietf-ace-oauth-authz].

This parameter is used to require and retrieve from the Group Manager information and parameters about the ECDH algorithm and about the public keys to be used in the OSCORE group to compute a static-static Diffie-Hellman shared secret [NIST-800-56A], in case the group supports the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

When used in the request, the 'ecdh_info' parameter encodes the CBOR simple value Null, to require information and parameters on the ECDH algorithm and on the public keys to be used to compute Diffie-Hellman shared secrets in the OSCORE group.

The CDDL notation [RFC8610] of the 'ecdh_info' parameter formatted as in the request is given below.

```
   ecdh_info_req = nil
```

The 'ecdh_info' parameter of the 2.01 (Created) response is a CBOR array of one or more elements. The number of elements is at most the number of OSCORE groups the client has been authorized to join.

Each element contains information about ECDH parameters and about public keys, for one or more OSCORE groups that support the pairwise mode of Group OSCORE and that the client has been authorized to join. Each element is formatted as follows.

- o The first element 'id' is the group name of the OSCORE group or an array of group names for the OSCORE groups for which the specified information applies.

- o The second element 'ecdh_alg' is an CBOR integer or a CBOR text string indicating the ECDH algorithm used in the OSCORE group identified by 'gname'. Values are taken from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms].

- o The third element 'ecdh_parameters' is a CBOR array indicating the parameters of the ECDH algorithm used in the OSCORE group identified by 'gname'. The CBOR array includes the following two elements, and its exact content depends on the value of the 'ecdh_alg' element.

  * 'ecdh_alg_capab', encoded as a CBOR array. Its format and value are the same of the COSE capabilities for the algorithm indicated in 'ecdh_alg', as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" Registry [COSE.Algorithms].

  * 'ecdh_key_type_capab', encoded as a CBOR array. Its format and value are the same of the COSE capabilities for the COSE key type of the keys used with the algorithm indicated in 'ecdh_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types].

- The fourth element 'ecdh_key_parameters' is a CBOR array indicating the parameters of the keys used with the ECDH algorithm in the OSCORE group identified by 'gname'. Its content depends on the value of 'ecdh_alg'. In particular, its format and value are the same of the COSE capabilities for the COSE key type of the keys used with the algorithm indicated in 'ecdh_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" Registry [COSE.Key.Types].

- The fifth element 'pub_key_enc' is CBOR integer indicating the encoding of public keys used in the OSCORE group identified by 'gname'. It takes value 1 ("COSE_Key") from the 'Confirmation Key' column of the "CWT Confirmation Method" Registry [CWT.Confirmation.Methods], so indicating that public keys in the OSCORE group are encoded as COSE Keys [I-D.ietf-cose-rfc8152bis-struct]. Future specifications may define additional values for this parameter.

The CDDL notation [RFC8610] of the 'ecdh_info' parameter formatted as in the response is given below.

```
      ecdh_info_res = [ + ecdh_info_entry ]

      ecdh_info_entry =
      [
        id : gname / [ + gname ],
        ecdh_alg : int / tstr,
        ecdh_parameters : [ any ],
        ecdh_key_parameters : [ any ],
        pub_key_enc = int
      ]

      gname = tstr
```

6.2. Sending the Joining Request

The joining node requests to join the OSCORE group by sending a Joining Request message to the related group-membership resource at the Group Manager, as per Section 4.3 of [I-D.ietf-ace-key-groupcomm].

Additionally to what defined in [I-D.ietf-ace-key-groupcomm], the following applies.

- The 'scope' parameter MUST be included. Its value encodes one scope entry with the format defined in Section 3, indicating the group name and the role(s) that the joining node wants to take in the group.

- The 'get_pub_keys' parameter is present only if the joining node wants to retrieve the public keys of the group members from the Group Manager during the joining process (see Section 7). Otherwise, this parameter MUST NOT be present.

  If this parameter is present and its value is non-null, each element of the first inner CBOR array is encoded as a CBOR unsigned integer, with the same value of a permission set ("Tperm") indicating that role or combination of roles in a scope entry, as defined in Section 3.

- 'cnonce' contains a dedicated nonce N_C generated by the joining node. For the N_C value, it is RECOMMENDED to use a 8-byte long random nonce.

- The signature encoded in the 'client_cred_verify' parameter is computed by the joining node by using the same private key and countersignature algorithm it intends to use for signing messages in the OSCORE group. Moreover, N_S is as defined in Section 6.2.1.

6.2.1. Value of the N_S Challenge

The value of the N_S challenge is determined as follows.

1. If the joining node has posted the Access Token to the /authz-info endpoint of the Group Manager as in Section 6.1, N_S takes the same value of the most recent 'kdcchallenge' parameter received by the joining node from the Group Manager. This can be either the one specified in the 2.01 (Created) response to the Token POST, or the one possibly specified in a 4.00 (Bad Request) response to a following Joining Request (see Section 6.3).

2. If the Token posting has relied on the DTLS profile of ACE [I-D.ietf-ace-dtls-authorize] with the Access Token as content of the "psk_identity" field of the ClientKeyExchange message [RFC6347], N_S is an exporter value computed as defined in Section 7.5 of [RFC8446]. Specifically, N_S is exported from the DTLS session between the joining node and the Group Manager, using an empty 'context_value', 32 bytes as 'key_length', and the exporter label "EXPORTER-ACE-Sign-Challenge-coap-group-oscore-app" defined in Section 21.6 of this specification.

It is up to applications to define how N_S is computed in further alternative settings.

Section 20.3 provides security considerations on the reusage of the N_S challenge.

6.3. Processing the Joining Request

The Group Manager processes the Joining Request as defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm]. Additionally, the following applies.

o The Group Manager MUST return a 5.03 (Service Unavailable) response in case the OSCORE group that the joining node has been trying to join is currently inactive (see Section 5.1).

o In case the joining node is not going to join the group exclusively as monitor and the Joining Request does not include the 'client_cred' parameter, the joining process fails if the Group Manager either: i) does not store a public key with an accepted format for the joining node; or ii) stores multiple public keys with an accepted format for the joining node.

o To compute the signature contained in 'client_cred_verify', the Group Manager considers:

   * as signed value, the value of the 'scope' parameter from the Joining Request as a CBOR byte string, concatenated with N_S encoded as a CBOR byte string, concatenated with N_C encoded as a CBOR byte string. In particular, N_S is determined as described in Section 6.2.1, while N_C is the nonce provided in the 'cnonce' parameter of the Joining Request;

   * the countersignature algorithm used in the OSCORE group, and possible correponding parameters;

   * the public key of the joining node, either retrieved from the 'client_cred' parameter, or already stored as acquired from previous interactions with the joining node.

- A 4.00 (Bad Request) response from the Group Manager to the joining node MUST have content format application/ace+cbor. The response payload is a CBOR map which MUST contain the 'sign_info' parameter, including a single element 'sign_info_entry' pertaining to the OSCORE group that the joining node has tried to join with the Joining Request. If the group supports the pairwise mode of Group OSCORE, the CBOR map MUST contain also the 'ecdh_info' parameter, including a single element 'ecdh_info_entry' pertaining to the OSCORE group that the joining node has tried to join with the Joining Request.

- The Group Manager MUST return a 4.00 (Bad Request) response in case the 'scope' parameter is not present in the Joining Request, or if it is present and specifies any set of roles not included in the following list: "requester", "responder", "monitor", ("requester", "responder"). Future specifications that define a new role MUST define possible sets of roles including the new one and existing ones, that are acceptable to specify in the 'scope' parameter of a Joining Request.

- The Group Manager MUST return a 4.00 (Bad Request) response in case the Joining Request includes the 'client_cred' parameter but does not include both the 'cnonce' and 'client_cred_verify' parameters.

- The Group Manager MUST return a 4.00 (Bad Request) response in case it cannot retrieve a public key with an accepted format for the joining node, either from the 'client_cred' parameter or as already stored.

- When receiving a 4.00 Bad Request response, the joining node SHOULD send a new Joining Request to the Group Manager, where:

* The ’cnonce’ parameter MUST include a new dedicated nonce N_C generated by the joining node.

* The ’client_cred’ parameter MUST include a public key compatible with the encoding, countersignature algorithm and possible associated parameters indicated by the Group Manager.

* The ’client_cred_verify’ parameter MUST include a signature computed as described in Section 6.2, by using the public key indicated in the current ’client_cred’ parameter, with the countersignature algorithm and possible associated parameters indicated by the Group Manager. If the error response from the Group Manager included the ’kdcchallenge’ parameter, the joining node MUST use its content as new N_S challenge to compute the signature.

6.4. Joining Response

If the processing of the Joining Request described in Section 6.3 is successful, the Group Manager updates the group membership by registering the joining node NODENAME as a new member of the OSCORE group GROUPNAME, as described in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm].

If the joining node has not taken exclusively the role of monitor, the Group Manager performs also the following actions.

- The Group Manager selects an available OSCORE Sender ID in the OSCORE group, and exclusively assigns it to the joining node. Consistently with Section 3 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MUST assign a Sender ID that has never been assigned before in the OSCORE group. The Group Manager MUST NOT assign a Sender ID to the joining node if this joins the group exclusively with the role of monitor, according to what specified in the Access Token (see Section 4.2).

- The Group Manager stores the association between i) the public key of the joining node; and ii) the Group Identifier (Gid), i.e. the OSCORE ID Context, associated to the OSCORE group together with the OSCORE Sender ID assigned to the joining node in the group. The Group Manager MUST keep this association updated over time.

Then, the Group Manager replies to the joining node, providing the updated security parameters and keying meterial necessary to participate in the group communication. This success Joining Response is formatted as defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], with the following additions:

o The 'gkty' parameter identifies a key of type "Group_OSCORE_Input_Material object", defined in Section 21.2 of this specification.

o The 'key' parameter includes what the joining node needs in order to set up the Group OSCORE Security Context as per Section 2 of [I-D.ietf-core-oscore-groupcomm].

   This parameter has as value a Group_OSCORE_Input_Material object, which is defined in this specification and extends the OSCORE_Input_Material object encoded in CBOR as defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile]. In particular, it contains the additional parameters 'group_senderId' 'cs_alg', 'cs_params', 'cs_key_params', 'cs_key_enc', 'ecdh_alg', 'ecdh_params' and 'ecdh_key_params' defined in Section 21.5 of this specification.

   More specifically, the 'key' parameter is composed as follows.

   * The 'ms' parameter MUST be present and includes the OSCORE Master Secret value used in the OSCORE group.

   * The 'hkdf' parameter, if present, has as value the KDF algorithm used in the OSCORE group.

   * The 'alg' parameter, if present, has as value the AEAD algorithm used in the OSCORE group.

   * The 'salt' parameter, if present, has as value the OSCORE Master Salt used in the OSCORE group.

   * The 'contextId' parameter MUST be present and has as value the Group Identifier (Gid), i.e. the OSCORE ID Context of the OSCORE group.

   * The 'group_senderId' parameter, if present, has as value the OSCORE Sender ID assigned to the joining node by the Group Manager, as described above. This parameter is not present if the node joins the group exclusively with the role of monitor, according to what specified in the Access Token (see Section 4.2). In any other case, this parameter MUST be present.

   * The 'cs_alg' parameter MUST be present and specifies the algorithm used to countersign messages in the OSCORE group. This parameter takes values from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms].

* The ’cs_params’ parameter MAY be present and specifies the parameters for the counter signature algorithm. This parameter is a CBOR array, which includes the following two elements:

  + ’sign_alg_capab’, with the same encoding as defined in Section 6.1. The value is the same as in the Token Post response where the ’sign_parameters’ value was non-null.

  + ’sign_key_type_capab’, with the same encoding as defined in Section 6.1. The value is the same as in the Token Post response where the ’sign_parameters’ value was non-null.

* The ’cs_key_params’ parameter MAY be present and specifies the parameters for the key used with the counter signature algorithm. This parameter is a CBOR array, with the same non-null encoding and value as ’sign_key_parameters’ of the Section 6.1.

* The ’cs_key_enc’ parameter MAY be present and specifies the encoding of the public keys of the group members. This parameter is a CBOR integer, whose value is 1 ("COSE_Key") taken from the ’Confirmation Key’ column of the "CWT Confirmation Method" Registry [CWT.Confirmation.Methods], so indicating that public keys in the OSCORE group are encoded as COSE Keys [I-D.ietf-cose-rfc8152bis-struct]. Future specifications may define additional values for this parameter. If this parameter is not present, 1 ("COSE_Key") MUST be assumed as default value.

* The ’ecdh_alg’ parameter, if present, specifies the ECDH algorithm used in the OSCORE group, if this supports the pairwise mode of Group OSCORE. This parameter takes values from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. This parameter MUST be present if the OSCORE group supports the pairwise mode of Group OSCORE, and MUST NOT be present otherwise.

* The ’ecdh_params’ parameter, if present, specifies the parameters for the ECDH algorithm. It MUST be present if the ’ecdh_alg’ parameter is present, and MUST NOT be present otherwise. This parameter is a CBOR array, which includes the following two elements:

  + ’ecdh_alg_capab’, with the same encoding as defined in Section 6.1.1. The value is the same as in the Token Post response where the ecdh_parameters’ value is non-null.

      + 'ecdh_key_type_capab', with the same encoding as defined in Section 6.1.1. The value is the same as in the Token Post response where the 'ecdh_parameters' value is non-null.

   * The 'ecdh_key_params' parameter, if present, specifies the parameters for the key used with the ECDH algorithm. It MUST be present if the 'ecdh_alg' parameter is present, and MUST NOT be present otherwise. This parameter is a CBOR array, with the same non-null encoding and value of 'ecdh_key_parameters' defined in Section 6.1.1.

- The 'exp' parameter MUST be present.

- The 'ace-groupcomm-profile' parameter MUST be present and has value coap_group_oscore_app (TBD3), which is defined in Section 21.3 of this specification.

- The 'pub_keys' parameter, if present, includes the public keys requested by the joining node by means of the 'get_pub_keys' parameter in the Joining Request. If public keys are encoded as COSE_Keys, each of them has as 'kid' the Sender ID that the corresponding owner has in the OSCORE group, thus used as group member identifier encoded as a CBOR byte string (REQ9).

  If the joining node has asked for the public keys of all the group members, i.e. 'get_pub_keys' had value Null in the Joining Request, then the Group Manager provides only the public keys of the group members that are relevant to the joining node. That is, in such a case, 'pub_keys' includes only: i) the public keys of the responders currently in the OSCORE group, in case the joining node is configured (also) as requester; and ii) the public keys of the requesters currently in the OSCORE group, in case the joining node is configured (also) as responder or monitor.

- The 'group_policies' parameter SHOULD be present, and SHOULD include the following elements:

   * "Sequence Number Synchronization Method" defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], with default value 1 ("Best effort");

   * "Key Update Check Interval" defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], with default value 3600;

   * "Expiration Delta" defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], with default value 0.

Finally, the joining node uses the information received in the Joining Response to set up the Group OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. In addition, the joining node maintains an association between each public key retrieved from the 'pub_keys' parameter and the role(s) that the corresponding group member has in the OSCORE group.

From then on, the joining node can exchange group messages secured with Group OSCORE as described in [I-D.ietf-core-oscore-groupcomm]. When doing so:

- o The joining node MUST NOT process an incoming request message, if protected by a group member whose public key is not associated to the role "Requester".
- o The joining node MUST NOT process an incoming response message, if protected by a group member whose public key is not associated to the role "Responder".
- o The joining node MUST NOT use the pairwise mode of Group OSCORE to process messages in the group, if the Joining Response did not include the 'ecdh_alg' parameter.

If the application requires backward security, the Group Manager MUST generate updated security parameters and group keying material, and provide it to the current group members upon the new node's joining (see Section 18). As a consequence, the joining node is not able to access secure communication in the OSCORE group occurred prior its joining.

7. Public Keys of Joining Nodes

Source authentication of a message sent within the group and protected with Group OSCORE is ensured by means of a digital counter signature embedded in the message (in group mode), or by integrity-protecting the message with pairwise keying material derived from the asymmetric keys of sender and recipient (in pairwise mode).

Therefore, group members must be able to retrieve each other's public key from a trusted key repository, in order to verify source authenticity of incoming group messages.

As also discussed in [I-D.ietf-core-oscore-groupcomm], the Group Manager acts as trusted repository of the public keys of the group members, and provides those public keys to group members if requested to. Upon joining an OSCORE group, a joining node is thus expected to provide its own public key to the Group Manager.

In particular, one of the following four cases can occur when a new node joins an OSCORE group.

o The joining node is going to join the group exclusively as monitor. That is, it is not going to send messages to the group, and hence to produce signatures with its own private key. In this case, the joining node is not required to provide its own public key to the Group Manager, which thus does not have to perform any check related to the public key encoding, or to a countersignature algorithm and possible associated parameters for that joining node. In case that joining node still provides a public key in the 'client_cred' parameter of the Joining Request (see Section 6.2), the Group Manager silently ignores that parameter, as well as the related parameters 'cnonce' and 'client_cred_verify'.

o The Group Manager already acquired the public key of the joining node during a past joining process. In this case, the joining node MAY choose not to provide again its own public key to the Group Manager, in order to limit the size of the Joining Request. The joining node MUST provide its own public key again if it has provided the Group Manager with multiple public keys during past joining processes, intended for different OSCORE groups. If the joining node provides its own public key, the Group Manager performs consistency checks as per Section 6.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.

o The joining node and the Group Manager use an asymmetric proof-of-possession key to establish a secure communication association. Then, two cases can occur.

   1. The proof-of-possession key is compatible with the encoding as well as with the counter signature algorithm and possible associated parameters used in the OSCORE group. Then, the Group Manager considers the proof-of-possession key as the public key associated to the joining node in the OSCORE group. If the joining node is aware that the proof-of-possession key is also valid for the OSCORE group, it MAY not provide it again as its own public key to the Group Manager. The joining node MUST provide its own public key again if it has provided the Group Manager with multiple public keys during past joining processes, intended for different OSCORE groups. If the joining node provides its own public key in the 'client_cred' parameter of the Joining Request (see Section 6.2), the Group Manager performs consistency checks as per Section 6.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.

2. The proof-of-possession key is not compatible with the encoding or with the counter signature algorithm and possible associated parameters used in the OSCORE group. In this case, the joining node MUST provide a different compatible public key to the Group Manager in the 'client_cred' parameter of the Joining Request (see Section 6.2). Then, the Group Manager performs consistency checks on this latest provided public key as per Section 6.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.

- The joining node and the Group Manager use a symmetric proof-of-possession key to establish a secure communication association. In this case, upon performing a joining process with that Group Manager for the first time, the joining node specifies its own public key in the 'client_cred' parameter of the Joining Request targeting the group-membership endpoint (see Section 6.2).

8. Retrieval of Updated Keying Material

At some point, a group member considers the Group OSCORE Security Context invalid and to be renewed. This happens, for instance, after a number of unsuccessful security processing of incoming messages from other group members, or when the Security Context expires as specified by the 'exp' parameter of the Joining Response.

When this happens, the group member retrieves updated security parameters and group keying material. This can occur in the two different ways described below.

8.1. Retrieval of Group Keying Material

If the group member wants to retrieve only the latest group keying material, it sends a Key Distribution Request to the Group Manager.

In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME at the Group Manager.

The Group Manager processes the Key Distribution Request according to Section 4.1.2.2 of [I-D.ietf-ace-key-groupcomm]. The Key Distribution Response is formatted as defined in Section 4.1.2.2 of [I-D.ietf-ace-key-groupcomm]. In addition:

- The 'key' parameter is formatted as defined in Section 6.4 of this specification, with the difference that it does not include the 'group_SenderId' parameter.

- The 'exp' parameter MUST be present.

o The 'ace-groupcomm-profile' parameter MUST be present and has value coap_group_oscore_app.

Upon receiving the Key Distribution Response, the group member retrieves the updated security parameters and group keying material, and, if they differ from the current ones, uses them to set up the new Group OSCORE Security Context as described in Section 2 of [I-D.ietf-core-oscore-groupcomm].

8.2. Retrieval of Group Keying Material and Sender ID

If the group member wants to retrieve the latest group keying material as well as the Sender ID that it has in the OSCORE group, it sends a Key Distribution Request to the Group Manager.

In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the Group Manager.

The Group Manager processes the Key Distribution Request according to Section 4.1.6.2 of [I-D.ietf-ace-key-groupcomm]. The Key Distribution Response is formatted as defined in Section 4.1.6.2 of [I-D.ietf-ace-key-groupcomm]. In addition:

o The 'key' parameter is formatted as defined in Section 6.4 of this specification. In particular, if the requesting group member has exclusively the role of monitor, no 'group_SenderId' is specified within the 'key' parameter.

   Note that, in any other case, the current Sender ID of the group member is not specified as a separate parameter, but rather specified as 'group_SenderId' within the 'key' parameter.

o The 'exp' parameter MUST be present.

Upon receiving the Key Distribution Response, the group member retrieves the updated security parameters, group keying material and Sender ID, and, if they differ from the current ones, uses them to set up the new Group OSCORE Security Context as described in Section 2 of [I-D.ietf-core-oscore-groupcomm].

9. Requesting a Change of Keying Material

As discussed in Section 2.4.2 of [I-D.ietf-core-oscore-groupcomm], a group member may at some point exhaust its Sender Sequence Numbers in the OSCORE group.

When this happens, the group member MUST send a Key Renewal Request message to the Group Manager, as per Section 4.5 of

[I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP PUT request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the Group Manager.

Upon receiving the Key Renewal Request, the Group Manager processes it as defined in Section 4.1.6.1 of [I-D.ietf-ace-key-groupcomm], and performs one of the following actions.

1. If the requesting group member has exclusively the role of monitor, the Group Manager replies with a 4.00 (Bad Request) error response.

2. Otherwise, the Group Manager takes one of the following actions.

   a. The Group Manager rekeys the OSCORE group. That is, the Group Manager generates new group keying material for that group (see Section 18), and replies to the group member with a group rekeying message as defined in Section 18, providing the new group keying material. Then, the Group Manager rekeys the rest of the OSCORE group, as discussed in Section 18.

   The Group Manager SHOULD perform a group rekeying only if already scheduled to occur shortly, e.g. according to an application-dependent rekeying period, or as a reaction to a recent change in the group membership. In any other case, the Group Manager SHOULD NOT rekey the OSCORE group when receiving a Key Renewal Request (OPT8).

   b. The Group Manager generates a new Sender ID for that group member and replies with a Key Renewal Response, formatted as defined in Section 4.1.6.1 of [I-D.ietf-ace-key-groupcomm]. In particular, the CBOR Map in the response payload includes a single parameter 'group_SenderId' defined in Section 21.1 of this document, specifying the new Sender ID of the group member encoded as a CBOR byte string.

   Consistently with Section 2.4.3.1 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MUST assign a new Sender ID that has never been assigned before in the OSCORE group.

10. Retrieval of Public Keys and Roles for Group Members

A group member or a signature verifier may need to retrieve the public keys of (other) group members. To this end, the group member or signature verifier sends a Public Key Request message to the Group Manager, as per Section 4.6 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends the request to the endpoint /ace-group/GROUPNAME/pub-key at the Group Manager.

If the Public Key Request uses the method FETCH, the Public Key Request is formatted as defined in Section 4.1.3.1 of [I-D.ietf-ace-key-groupcomm]. In particular:

o Each element (if any) of the first inner CBOR array is formatted as in the first inner CBOR array of the 'get_pub_keys' parameter of the Joining Request when the parameter value is non-null (see Section 6.2).

o Each element (if any) of the second inner CBOR array is a CBOR byte string (REQ9), which encodes the Sender ID of the group member for which the associated public key is requested.

Upon receiving the Public Key Request, the Group Manager processes it as per Section 4.1.3.1 or 4.1.3.2 of [I-D.ietf-ace-key-groupcomm], depending on the request method being FETCH or GET, respectively. Additionally, if the Public Key Request uses the method FETCH, the Group Manager silently ignores node identifiers included in the 'get_pub_keys' parameter of the request that are not associated to any current group member.

The success Public Key Response is formatted as defined in Section 4.1.3.1 or 4.1.3.2 of [I-D.ietf-ace-key-groupcomm], depending on the request method being FETCH or GET, respectively.

11. Update of Public Key

A group member may need to provide the Group Manager with its new public key to use in the group from then on, hence replacing the current one. This can be the case, for instance, if the countersignature algorithm and possible associated parameters used in the OSCORE group have been changed, and the current public key is not compatible with them.

To this end, the group member sends a Public Key Update Request message to the Group Manager, as per Section 4.7 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP POST request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME/pub-key at the Group Manager.

Upon receiving the Group Leaving Request, the Group Manager processes it as per Section 4.1.7.1 of [I-D.ietf-ace-key-groupcomm], with the following additions.

- If the requesting group member has exclusively the role of monitor, the Group Manager replies with a 4.00 (Bad request) error response.

- The N_S signature challenge is computed as per point (1) in Section 6.2.1 (REQ17).

- If the request is successfully processed, the Group Manager stores the association between i) the new public key of the group member; and ii) the Group Identifier (Gid), i.e. the OSCORE ID Context, associated to the OSCORE group together with the OSCORE Sender ID assigned to the group member in the group. The Group Manager MUST keep this association updated over time.

12. Retrieval of Group Policies

A group member may request the current policies used in the OSCORE group. To this end, the group member sends a Policies Request, as per Section 4.8 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/policies at the Group Manager, where GROUPNAME is the name of the OSCORE group.

Upon receiving the Policies Request, the Group Manager processes it as per Section 4.1.4.1 of [I-D.ietf-ace-key-groupcomm]. The success Policies Response is formatted as defined in Section 4.1.4.1 of [I-D.ietf-ace-key-groupcomm].

13. Retrieval of Keying Material Version

A group member may request the current version of the keying material used in the OSCORE group. To this end, the group member sends a Version Request, as per Section 4.9 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/num at the Group Manager, where GROUPNAME is the name of the OSCORE group.

Upon receiving the Version Request, the Group Manager processes it as per Section 4.1.5.1 of [I-D.ietf-ace-key-groupcomm]. The success Version Response is formatted as defined in Section 4.1.5.1 of [I-D.ietf-ace-key-groupcomm].

14. Retrieval of Group Status

A group member may request the current status of the the OSCORE group, i.e. active or inactive. To this end, the group member sends a Group Status Request to the Group Manager.

In particular, the group member sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/active at the Group Manager defined in Section 5.1 of this specification, where GROUPNAME is the name of the OSCORE group. The success Group Version Response is formatted as defined in Section 5.1 of this specification.

Upon learning from a 2.05 (Content) response that the group is currently inactive, the group member SHOULD stop taking part in communications within the group, until it becomes active again.

Upon learning from a 2.05 (Content) response that the group has become active again, the group member can resume taking part in communications within the group.

Figure 3 gives an overview of the exchange described above.

```
 Group                                                          Group
 Member                                                       Manager
   |                                                              |
   |--- Group Status Request: GET ace-group/GROUPNAME/active --->|
   |                                                              |
   |<---------- Group Status Response: 2.05 (Content) -----------|
   |                                                              |
```

Figure 3: Message Flow of Group Status Request-Response

15. Retrieval of Group Names and URIs

A node may want to retrieve from the Group Manager the group name and the URI of the group-membership resource of a group. This is relevant in the following cases.

- o Before joining a group, a joining node may know only the current Group Identifier (Gid) of that group, but not the group name and the URI to the group-membership resource.

- o As current group member in several groups, the node has missed a previous group rekeying in one of them (see Section 18). Hence, it retains stale keying material and fails to decrypt received messages exchanged in that group.

  Such messages do not provide a direct hint to the correct group name, that the node would need in order to retrieve the latest keying material and public keys from the Group Manager (see Section 8.1, Section 8.2 and Section 10). However, such messages may specify the current Gid of the group, as value of the 'kid_context' field of the OSCORE CoAP option (see Section 6.1 of [RFC8613] and Section 4.2 of [I-D.ietf-core-oscore-groupcomm]).

o As signature verifier, the node also refers to a group name for retrieving the required public keys from the Group Manager (see Section 10). As discussed above, intercepted messages do not provide a direct hint to the correct group name, while they may specify the current Gid of the group, as value of the 'kid_context' field of the OSCORE CoAP option. In such a case, upon intercepting a message in the group, the node requires to correctly map the Gid currently used in the group with the invariant group name.

   Furthermore, since it is not a group member, the node does not take part to a possible group rekeying. Thus, following a group rekeying and the consequent change of Gid in a group, the node would retain the old Gid value and cannot correctly associate intercepted messages to the right group, especially if acting as signature verifier in several groups. This in turn prevents the efficient verification of counter signatures, and especially the retrieval of required, new public keys from the Group Manager.

In either case, the node only knows the current Gid of the group, as learnt from received or intercepted messages exchanged in the group. As detailed below, the node can contact the Group Manager, and request the group name and URI to the group-membership resource corresponding to that Gid. Then, it can use that information to either join the group as a candidate group member, get the latest keying material as a current group member, or retrieve public keys used in the group as a signature verifier. To this end, the node sends a Group Name and URI Retrieval Request, as per Section 4.2 of [I-D.ietf-ace-key-groupcomm].

In particular, the node sends a CoAP FETCH request to the endpoint /ace-group at the Group Manager formatted as defined in Section 4.1.1.1 of [I-D.ietf-ace-key-groupcomm]. Each element of the CBOR array 'gid' is a CBOR byte string (REQ7b), which encodes the Gid of the group for which the group name and the URI to the group-membership resource are requested.

Upon receiving the Group Name and URI Retrieval Request, the Group Manager processes it as per Section 4.1.1.1 of [I-D.ietf-ace-key-groupcomm]. The success Group Name and URI Retrieval Response is formatted as defined in Section 4.1.1.1 of [I-D.ietf-ace-key-groupcomm]. In particular, each element of the CBOR array 'gid' is a CBOR byte string (REQ7b), which encodes the Gid of the group for which the group name and the URI to the group-membership resource are provided.

For each of its groups, the Group Manager maintains an association between the group name and the URI to the group-membership resource on one hand, and only the current Gid for that group on the other hand. That is, the Group Manager MUST NOT maintain an association between the former pair and any other Gid for that group than the current, most recent one.

Figure 4 gives an overview of the exchanges described above.

```
                                                                 Group
  Node                                                          Manager
   |                                                                |
   |---- Group Name and URI Retrieval Request: FETCH ace-group/ --->|
   |                                                                |
   |<--- Group Name and URI Retrieval Response: 2.05 (Content) -----|
   |                                                                |
```

Figure 4: Message Flow of Group Name and URI Retrieval Request-Response

16. Request to Leave the Group

A group member may request to leave the OSCORE group. To this end, the group member sends a Group Leaving Request, as per Section 4.10 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP DELETE request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the Group Manager.

Upon receiving the Group Leaving Request, the Group Manager processes it as per Section 4.1.6.3 of [I-D.ietf-ace-key-groupcomm].

17. Removal of a Group Member

Other than after a spontaneous request to the Group Manager as described in Section 16, a node may be forcibly removed from the OSCORE group, e.g. due to expired or revoked authorization.

If, upon joining the group (see Section 6.2), the leaving node specified a URI in the 'control_path' parameter defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], the Group Manager MUST inform the leaving node of its eviction, by sending a DELETE request targeting the URI specified in the 'control_path' parameter (OPT9).

If the leaving node has not exclusively the role of monitor, the Group Manager performs the following actions.

o The Group Manager frees the OSCORE Sender ID value of the leaving node. However, this value MUST NOT become available for possible upcoming joining nodes in the same group.

- The Group Manager cancels the association between, on one hand, the public key of the leaving node and, on the other hand, the Group Identifier (Gid) associated to the OSCORE group together with the freed OSCORE Sender ID value. The Group Manager deletes the public key of the leaving node, if that public key has no remaining association with any pair (Gid, Sender ID).

If the application requires forward security, the Group Manager MUST generate updated security parameters and group keying material, and provide it to the remaining group members (see Section 18). As a consequence, the leaving node is not able to acquire the new security parameters and group keying material distributed after its leaving.

Same considerations in Section 5 of [I-D.ietf-ace-key-groupcomm] apply here as well, considering the Group Manager acting as KDC.

18. Group Rekeying Process

In order to rekey the OSCORE group, the Group Manager distributes a new Group Identifier (Gid), i.e. a new OSCORE ID Context; a new OSCORE Master Secret; and, optionally, a new OSCORE Master Salt for that group. When doing so, the Group Manager MUST increment the version number of the group keying material, before starting its distribution.

Consistently with Section 2.3 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MUST assign a Gid that has never been assigned before to the OSCORE group.

Furthermore, the Group Manager MUST preserve the same unchanged Sender IDs for all group members. This avoids affecting the retrieval of public keys from the Group Manager as well as the verification of group messages.

The Group Manager MUST support at least the following group rekeying scheme. Future application profiles may define alternative message formats and distribution schemes.

As group rekeying message, the Group Manager uses the same format of the Joining Response message in Section 6.4. In particular:

- Only the parameters 'gkty', 'key', 'num', 'ace-groupcomm-profile' and 'exp' are present.

- The 'ms' parameter of the 'key' parameter specifies the new OSCORE Master Secret value.

- The 'contextId' parameter of the 'key' parameter specifies the new Group ID used as OSCORE ID Context value.

The Group Manager separately sends a group rekeying message to each group member to be rekeyed.

Each rekeying message MUST be secured with the pairwise secure communication channel between the Group Manager and the group member used during the joining process. In particular, each rekeying message can target the 'control_path' URI path defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm] (OPT9), if provided by the intended recipient upon joining the group (see Section 6.2).

It is RECOMMENDED that the Group Manager gets confirmation of successful distribution from the group members, and admits a maximum number of individual retransmissions to non-confirming group members.

This approach requires group members to act (also) as servers, in order to correctly handle unsolicited group rekeying messages from the Group Manager. In particular, if a group member and the Group Manager use OSCORE [RFC8613] to secure their pairwise communications, the group member MUST create a Replay Window in its own Recipient Context upon establishing the OSCORE Security Context with the Group Manager, e.g. by means of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

Group members and the Group Manager SHOULD additionally support alternative rekeying approaches that do not require group members to act (also) as servers. A number of such approaches are defined in Section 4.4 of [I-D.ietf-ace-key-groupcomm]. In particular, a group member may subscribe for updates to the group-membership resource of the group, at the endpoint /ace-group/GROUPNAME/ of the Group Manager. This can rely on CoAP Observe [RFC7641] or on a full-fledged Pub-Sub model [I-D.ietf-core-coap-pubsub] with the Group Manager acting as Broker.

In case the rekeying terminates and some group members have not received the new keying material, they will not be able to correctly process following secured messages exchanged in the group. These group members will eventually contact the Group Manager, in order to retrieve the current keying material and its version.

Some of these group members may be in multiple groups, each associated to a different Group Manager. When failing to correctly process messages secured with the new keying material, these group members may not have sufficient information to determine which exact Group Manager they should contact, in order to retrieve the current keying material they are missing.

If the Gid is formatted as described in Appendix C of [I-D.ietf-core-oscore-groupcomm], the Group Prefix can be used as a hint to determine the right Group Manager, as long as no collisions among Group Prefixes are experienced. Otherwise, a group member needs to contact the Group Manager of each group, e.g. by first requesting only the version of the current group keying material (see Section 13) and then possibly requesting the current keying material (see Section 8.1).

Furthermore, some of these group members can be in multiple groups, all of which associated to the same Group Manager. In this case, these group members may also not have sufficient information to determine which exact group they should refer to, when contacting the right Group Manager. Hence, they need to contact a Group Manager multiple times, i.e. separately for each group they belong to and associated to that Group Manager.

19. Default Values for Group Configuration Parameters

This section defines the default values that the Group Manager assumes for the configuration parameters of an OSCORE group, unless differently specified when creating and configuring the group. This can be achieved as specified in [I-D.ietf-ace-oscore-gm-admin].

The Group Manager SHOULD use the same default values defined in Section 3.2 of [RFC8613] for both the HKDF algorithm and the AEAD algorithm used in the group.

The Group Manager SHOULD use the following default values for the algorithm, algorithm parameters and key parameters used to countersign messages in the group, consistently with the "COSE Algorithms" Registry [COSE.Algorithms], the "COSE Key Types" Registry [COSE.Key.Types] and the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].

o For the algorithm 'cs_alg' used to countersign messages in the group, the signature algorithm EdDSA [RFC8032].

o For the parameters 'cs_params' of the counter signature algorithm:

  * The array [[OKP], [OKP, Ed25519]], indicating the elliptic curve Ed25519 [RFC8032], in case EdDSA is assumed or specified for 'cs_alg'.

  * The array [[EC2], [EC2, P-256]], indicating the elliptic curve P-256, in case ES256 [RFC6979] is specified for 'cs_alg'.

      * The array [[EC2], [EC2, P-384]], indicating the elliptic curve P-384, in case ES384 [RFC6979] is specified for 'cs_alg'.

      * The array [[EC2], [EC2, P-521]], indicating the elliptic curve P-521, in case ES512 [RFC6979] is specified for 'cs_alg'.

      * The array [[], [RSA]], in case PS256, PS384 or PS512 [RFC8017] is specified for 'cs_alg'.

   o For the parameters 'cs_key_params' of the key used with the counter signature algorithm:

      * The array [OKP, Ed25519] as pair (key type, elliptic curve), in case EdDSA is assumed or specified for 'cs_alg' and Ed25519 is assumed or specified within the second array of 'cs_params'.

      * The array [OKP, Ed448] as pair (key type, elliptic curve), in case EdDSA is assumed or specified for 'cs_alg' and the elliptic curve Ed448 [RFC8032] is specified within the second array of 'cs_params'.

      * The array [EC2, P-256] as pair (key type, elliptic curve), in case ES256 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-256 is assumed or specified within the second array of 'cs_params'.

      * The array [EC2, P-384] as pair (key type, elliptic curve), in case ES384 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-384 is specified within the second array of 'cs_params'.

      * The array [EC2, P-521] as pair (key type, elliptic curve), in case ES512 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-521 is specified within the second array of 'cs_params'.

      * The array [RSA] indicating RSA as key type, in case PS256, PS384 or PS512 [RFC8017] is specified for 'cs_alg'.

   o For the 'cs_key_enc' encoding of the public keys of the group members, COSE_Key from the "CWT Confirmation Methods" Registry [CWT.Confirmation.Methods].

If the group supports the pairwise mode of Group OSCORE, the Group Manager SHOULD use the following default values for the algorithm, algorithm parameters and key parameters used to compute static-static Diffie-Hellman shared secrets, consistently with the "COSE Algorithms" Registry [COSE.Algorithms], the "COSE Key Types" Registry

[COSE.Key.Types] and the "COSE Elliptic Curves" Registry [COSE.Elliptic.Curves].

- o For the algorithm 'ecdh_alg' used to compute static-static Diffie-Hellman shared secrets, the ECDH algorithm ECDH-SS + HKDF-256 specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs].

- o For the parameters 'ecdh_params' of the ECDH algorithm:

   * The array [[OKP], [OKP, X25519]], indicating the elliptic curve X25519 [RFC8032], in case EdDSA is assumed or specified for 'cs_alg'.

   * The array [[EC2], [EC2, P-256]], indicating the elliptic curve P-256, in case ES256 [RFC6979] is specified for 'cs_alg'.

   * The array [[EC2], [EC2, P-384]], indicating the elliptic curve P-384, in case ES384 [RFC6979] is specified for 'cs_alg'.

   * The array [[EC2], [EC2, P-521]], indicating the elliptic curve P-521, in case ES512 [RFC6979] is specified for 'cs_alg'.

- o For the parameters 'ecdh_key_params' of the key used with the ECDH algorithm:

   * The array [OKP, X25519] as pair (key type, elliptic curve), in case EdDSA is assumed or specified for 'cs_alg' and X25519 is assumed or specified within the second array of 'ecdh_params'.

   * The array [OKP, X448] as pair (key type, elliptic curve), in case EdDSA is assumed or specified for 'cs_alg' and the elliptic curve X448 [RFC8032] is specified within the second array of 'ecdh_params'.

   * The array [EC2, P-256] as pair (key type, elliptic curve), in case ES256 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-256 is assumed or specified within the second array of 'ecdh_params'.

   * The array [EC2, P-384] as pair (key type, elliptic curve), in case ES384 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-384 is specified within the second array of 'ecdh_params'.

   * The array [EC2, P-521] as pair (key type, elliptic curve), in case ES512 [RFC6979] is specified for 'cs_alg' and the elliptic curve P-521 is specified within the second array of 'ecdh_params'.

20. Security Considerations

Security considerations for this profile are inherited from [I-D.ietf-ace-key-groupcomm], the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the specific transport profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

The following security considerations also apply for this profile.

20.1. Management of OSCORE Groups

This profile leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

o Management of group keying material (see Section 3.1 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager is responsible for the renewal and re-distribution of the keying material in the groups of its competence (rekeying). According to the specific application requirements, this can include rekeying the group upon changes in its membership. In particular, renewing the group keying material is required upon a new node's joining or a current node's leaving, in case backward security and forward security have to be preserved, respectively.

o Provisioning and retrieval of public keys (see Section 2 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager acts as key repository of public keys of group members, and provides them upon request.

o Synchronization of sequence numbers (see Section 6.1 of [I-D.ietf-core-oscore-groupcomm]). This concerns how a responder node that has just joined an OSCORE group can synchronize with the sequence number of requesters in the same group.

Before sending the Joining Response, the Group Manager MUST verify that the joining node actually owns the associated private key. To this end, the Group Manager can rely on the proof-of-possession challenge-response defined in Section 6. Alternatively, the joining node can use its own public key as asymmetric proof-of-possession key to establish a secure channel with the Group Manager, e.g. as in Section 3.2 of [I-D.ietf-ace-dtls-authorize]. However, this requires such proof-of-possession key to be compatible with the encoding as well as with the countersignature algorithm and possible associated parameters used in the OSCORE group.

A node may have joined multiple OSCORE groups under different non-synchronized Group Managers. Therefore, it can happen that those OSCORE groups have the same Group Identifier (Gid). It follows that, upon receiving a Group OSCORE message addressed to one of those groups, the node would have multiple Security Contexts matching with the Gid in the incoming message. It is up to the application to decide how to handle such collisions of Group Identifiers, e.g. by trying to process the incoming message using one Security Context at the time until the right one is found.

20.2. Size of Nonces for Signature Challenge

With reference to the Joining Request message in Section 6.2, the proof-of-possession signature included in 'client_cred_verify' is computed over the challenge N_C | N_S, where | denotes concatenation.

For the N_C challenge share, it is RECOMMENDED to use a 8-byte long random nonce. Furthermore, N_C is always conveyed in the 'cnonce' parameter of the Joining Request, which is always sent over the secure communication channel between the joining node and the Group Manager.

As defined in Section 6.2.1, the way the N_S value is computed depends on the particular way the joining node provides the Group Manager with the Access Token, as well as on following interactions between the two.

- o If the Access Token is not explicitly posted to the /authz-info endpoint of the Group Manager, then N_S is computed as a 32-byte long challenge share. For an example, see point (2) of Section 6.2.1.

- o If the Access Token has been explicitly posted to the /authz-info endpoint of the Group Manager, N_S takes the most recent value provided to the client by the Group Manager in the 'kdcchallenge' parameter, as specified in point (1) of Section 6.2.1. This is provided either in the 2.01 response to the Token Post (see Section 6.1), or in a 4.00 response to a following Joining Request (see Section 6.3). In either case, it is RECOMMENDED to use a 8-byte long random challenge as value for N_S.

If we consider both N_C and N_S to take 8-byte long values, the following considerations hold.

- o Let us consider both N_C and N_S as taking random values, and the Group Manager to never change the value of the N_S provided to a Client during the lifetime of an Access Token. Then, as per the birthday paradox, the average collision for N_S will happen after

2^32 new posted Access Tokens, while the average collision for N_C will happen after 2^32 new Joining Requests. This amounts to considerably more token provisionings than the expected new joinings of OSCORE groups under a same Group Manager, as well as to considerably more requests to join OSCORE groups from a same Client using a same Access Token under a same Group Manager.

- Section 7 of [I-D.ietf-ace-oscore-profile] as well Appendix B.2 of [RFC8613] recommend the use of 8-byte random values as well. Unlike in those cases, the values of N_C and N_S considered in this specification are not used for as sensitive operations as the derivation of a Security Context, and thus do not have possible implications in the security of AEAD ciphers.

20.3. Reusage of Nonces for Signature Challenge

As long as the Group Manager preserves the same N_S value currently associated to an Access Token, i.e. the latest value provided to a Client in a 'kdcchallenge' parameter, the Client is able to successfully reuse the same signature challenge for multiple Joining Requests to that Group Manager.

In particular, the Client can reuse the same N_C value for every Joining Request to the Group Manager, and combine it with the same unchanged N_S value. This results in reusing the same signature challenge for producing the signature to include in the 'client_cred_verify' parameter of the Joining Requests.

Unless the Group Manager maintains a list of N_C values already used by that Client since the latest update to the N_S value associated to the Access Token, the Group Manager can be forced to falsely believe that the Client possesses its own private key at that point in time, upon verifying the signature in the 'client_cred_verify' parameter.

21. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[This specification]]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

21.1. ACE Groupcomm Parameters Registry

IANA is asked to register the following entry to the "ACE Groupcomm Parameters" Registry defined in Section 8.5 of [I-D.ietf-ace-key-groupcomm].

- Name: group_senderId
- CBOR Key: TBD1
- CBOR Type: Byte string
- Reference: [[This specification]] (Section 9)

21.2. ACE Groupcomm Key Registry

IANA is asked to register the following entry to the "ACE Groupcomm Key" Registry defined in Section 8.6 of [I-D.ietf-ace-key-groupcomm].

- Name: Group_OSCORE_Input_Material object
- Key Type Value: TBD2
- Profile: "coap_group_oscore_app", defined in Section 21.3 of this specification.
- Description: A Group_OSCORE_Input_Material object encoded as described in Section 6.4 of this specification.
- Reference: [[This specification]] (Section 6.4)

21.3. ACE Groupcomm Profile Registry

IANA is asked to register the following entry to the "ACE Groupcomm Profile" Registry defined in Section 8.7 of [I-D.ietf-ace-key-groupcomm].

- Name: coap_group_oscore_app
- Description: Application profile to provision keying material for participating in group communication protected with Group OSCORE as per [I-D.ietf-core-oscore-groupcomm].
- CBOR Value: TBD3
- Reference: [[This specification]] (Section 6.4)

21.4. Sequence Number Synchronization Method Registry

IANA is asked to register the following entries in the "Sequence Number Synchronization Method" Registry defined in Section 8.9 of [I-D.ietf-ace-key-groupcomm].

- Name: Best effort

- Value: 1

- Description: No action is taken.

- Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.1)

- Name: Baseline

- Value: 2

- Description: The first received request sets the baseline reference point, and is discarded with no delivery to the application.

- Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.2)

- Name: Echo challenge-response

- Value: 3

- Description: Challenge response using the Echo Option for CoAP from [I-D.ietf-core-echo-request-tag].

- Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.3)

21.5. OSCORE Security Context Parameters Registry

IANA is asked to register the following entries in the "OSCORE Security Context Parameters" Registry defined in Section 9.4 of [I-D.ietf-ace-oscore-profile].

- Name: group_SenderId

- CBOR Label: TBD4

- CBOR Type: bstr

- Registry: -

- Description: OSCORE Sender ID assigned to a member of an OSCORE group

- Reference: [[This specification]] (Section 6.4)

- Name: cs_alg

- CBOR Label: TBD5
- CBOR Type: tstr / int
- Registry: COSE Algorithms
- Description: OSCORE Counter Signature Algorithm Value
- Reference: [[This specification]] (Section 6.4)

- Name: cs_params
- CBOR Label: TBD6
- CBOR Type: array
- Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- Description: OSCORE Counter Signature Algorithm Additional Parameters
- Reference: [[This specification]] (Section 6.4)

- Name: cs_key_params
- CBOR Label: TBD7
- CBOR Type: array
- Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- Description: OSCORE Counter Signature Key Additional Parameters
- Reference: [[This specification]] (Section 6.4)

- Name: cs_key_enc
- CBOR Label: TBD8
- CBOR Type: integer
- Registry: CWT Confirmation Methods
- Description: Encoding of Public Keys to be used with the OSCORE Counter Signature Algorithm

- Reference: [[This specification]] (Section 6.4)

- Name: ecdh_alg
- CBOR Label: TBD9
- CBOR Type: tstr / int
- Registry: COSE Algorithms
- Description: OSCORE ECDH Algorithm Value
- Reference: [[This specification]] (Section 6.4)

- Name: ecdh_params
- CBOR Label: TBD10
- CBOR Type: array
- Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- Description: OSCORE ECDH Algorithm Additional Parameters
- Reference: [[This specification]] (Section 6.4)

- Name: ecdh_key_params
- CBOR Label: TBD11
- CBOR Type: array
- Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- Description: OSCORE ECDH Key Additional Parameters
- Reference: [[This specification]] (Section 6.4)

21.6. TLS Exporter Label Registry

IANA is asked to register the following entry to the "TLS Exporter Label" Registry defined in Section 6 of [RFC5705] and updated in Section 12 of [RFC8447].

- Value: EXPORTER-ACE-Sign-Challenge-coap-group-oscore-app

- DTLS-OK: Y

- Recommended: N

- Reference: [[This specification]] (Section 6.2.1)

21.7. AIF Registry

IANA is asked to register the following entry to the "Toid" sub-registry of the "AIF" Registry defined in Section 5.2 of [I-D.ietf-ace-aif].

- Name: oscore-group-name

- Description/Specification: group name of the OSCORE group, as specified in [[This specification]].

IANA is asked to register the following entry to the "Tperm" sub-Registry of the "AIF" Registry defined in Section 5.2 of [I-D.ietf-ace-aif].

- Name: oscore-group-roles

- Description/Specification: role(s) of the member of the OSCORE group, as specified in [[This specification]].

21.8. Media Type Registrations

This specification registers the 'application/aif-groupcomm-oscore+cbor' media type for the AIF specific data model AIF-OSCORE-GROUPCOMM defined in Section 3 of [[This specification]]. This registration follows the procedures specified in [RFC6838].

These media type has parameters for specifying the object identifier ("Toid") and set of permissions ("Tperm") defined for the AIF-generic model in [I-D.ietf-ace-aif]; default values are the values "oscore-group-name" for "Toid" and "oscore-group-roles" for "Tperm".

Type name: application

Subtype name: aif-groupcomm-oscore+cbor

Required parameters: "Toid", "Tperm"

Optional parameters: none

Encoding considerations: Must be encoded as a CBOR array, each element of which is an array [Toid, Tperm] as defined in Section 3 of [[This specification]].

Security considerations: See Section 20 of [[This specification]].

Interoperability considerations: n/a

Published specification: [[This specification]]

Applications that use this media type: The type is used by applications that want to express authorization information about joining OSCORE groups, as specified in [[This specification]].

Additional information: n/a

Person & email address to contact for further information: iesg@ietf.org [1]

Intended usage: COMMON

Restrictions on usage: None

Author: Marco Tiloca marco.tiloca@ri.se [2]

Change controller: IESG

21.9. CoAP Content-Format Registry

IANA is asked to register the following entry to the "CoAP Content-Formats" registry, within the "CoRE Parameters" registry:

Media Type: application/aif-groupcomm-oscore+cbor;Toid="oscore-group-name",Tperm"oscore-group-roles"

Encoding: -

ID: TBD12

Reference: [[This specification]]

21.10. Group OSCORE Roles Registry

This specification establishes the IANA "Group OSCORE Roles" Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 21.12.

This registry includes the possible roles that nodes can take in an OSCORE group, each in combination with a numeric identifier. These numeric identifiers are used to express authorization information about joining OSCORE groups, as specified in Section 3 of [[This specification]].

The columns of this registry are:

- Name: A value that can be used in documents for easier comprehension, to identify a possible role that nodes can take in an OSCORE group.

- Value: The numeric identifier for this role. Integer values greater than 65535 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].

- Description: This field contains a brief description of the role.

- Reference: This contains a pointer to the public specification for the role.

This registry will be initially populated by the values in Figure 1.

The Reference column for all of these entries will be [[This specification]].

21.11. CoRE Resource Type Registry

IANA is asked to register a new Resource Type (rt=) Link Target Attribute in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained Restful Environments (CoRE) Parameters" [CORE.Parameters] registry.

- Value: "core.osc.gm"

- Description: Group-membership resource of an OSCORE Group Manager.

- Reference: [[This specification]]

21.12. Expert Review Instructions

The IANA Registry established in this document is defined as "Expert Review". 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:

- Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Experts should inspect the entry for the considered role, to verify the correctness of its description against the role as intended in the specification that defined it. Expert should consider requesting an opinion on the correctness of registered parameters from the Authentication and Authorization for Constrained Environments (ACE) Working Group and the Constrained RESTful Environments (CoRE) Working Group.

  Entries that do not meet these objective of clarity and completeness should not be registered.

- Duplicated registration and 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.

- Experts should take into account the expected usage of roles when approving point assignment. Given a 'Value' V as code point, the length of the encoding of (2^(V+1) - 1) should be weighed against the usage of the entry, considering the resources and capabilities of devices it will be used on. Additionally, given a 'Value' V as code point, the length of the encoding of (2^(V+1) - 1) should be weighed against how many code points resulting in that encoding length are left, and the resources and capabilities of devices it will be used on.

- Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

22. References

22.1. Normative References

[CORE.Parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>.

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

[COSE.Elliptic.Curves]
IANA, "COSE Elliptic Curves", <https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>.

[COSE.Key.Types]
IANA, "COSE Key Types", <https://www.iana.org/assignments/cose/cose.xhtml#key-type>.

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

[I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-ietf-ace-aif-00 (work in progress), July 2020.

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-10 (work in progress), November 2020.

[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-35 (work in progress), June 2020.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-13 (work in progress), October 2020.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[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>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

[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>.

22.2. Informative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-14 (work in progress), October 2020.

[I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Hoeglund, R., Stok, P., Palombini, F., and K. Hartke, "Admin Interface for the OSCORE Group Manager", draft-ietf-ace-oscore-gm-admin-01 (work in progress), November 2020.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-09 (work in progress), September 2019.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-10 (work in progress), July 2020.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-07 (work in progress), November 2020.

[NIST-800-56A]
Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>.

[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>.

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

[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>.

[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>.

22.3. URIs

[1] mailto:iesg@ietf.org

[2] mailto:marco.tiloca@ri.se

Appendix A. Profile Requirements

This appendix lists the specifications on this application profile of ACE, based on the requirements defined in Appendix A of [I-D.ietf-ace-key-groupcomm].

- REQ1 - If the value of the GROUPNAME URI path and the group name in the Access Token scope (gname in Section 3.1 of [I-D.ietf-ace-key-groupcomm]) do not match, specify the mechanism to map the GROUPNAME value in the URI to the group name: not applicable, since a match is required.

- REQ2 - Specify the encoding and value of roles, for scope entries of 'scope': see Section 3 and Section 4.1.

- REQ3 - if used, specify the acceptable values for 'sign_alg': values from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms].

- REQ4 - If used, specify the acceptable values for 'sign_parameters': format and values from the COSE algorithm capabilities as specified in the "COSE Algorithms" Registry [COSE.Algorithms] and from the COSE key type capabilities as specified in the "COSE Key Types" Registry [COSE.Key.Types].

- REQ5 - If used, specify the acceptable values for 'sign_key_parameters': format and values from the COSE key type capabilities as specified in the "COSE Key Types" Registry [COSE.Key.Types].

- REQ6 - If used, specify the acceptable values for 'pub_key_enc': 1 ("COSE_Key") from the 'Confirmation Key' column of the "CWT Confirmation Method" Registry [CWT.Confirmation.Methods]. Future specifications may define additional values for this parameter.

- REQ7a - Register a Resource Type for the root url-path, which is used to discover the correct url to access at the KDC (see Section 4.1 of [I-D.ietf-ace-key-groupcomm]): the Resource Type (rt=) Link Target Attribute value "core.osc.gm" is registered in Section 21.11.

- REQ7aa - Define what operations (i.e. CoAP methods) are allowed on each resource, for each role defined in REQ2: see Section 5.2.

- REQ7b - Specify the exact encoding of group identifier (see Section 4.1.1.1 of [I-D.ietf-ace-key-groupcomm]): CBOR byte string (see Section 15).

- REQ7 - Format of the 'key' value: see Section 6.4.

- REQ8 - Acceptable values of 'gkty': Group_OSCORE_Input_Material object (see Section 6.4).

- o REQ9 - Specify the format of the identifiers of group members: CBOR byte string (see Section 6.4 and Section 10).

- o REQ10 - Specify the communication protocol that the members of the group must use: CoAP [RFC7252], possibly over IP multicast [I-D.ietf-core-groupcomm-bis].

- o REQ11 - Specify the security protocols that the group members must use to protect their communication: Group OSCORE [I-D.ietf-core-oscore-groupcomm].

- o REQ12 - Specify and register the application profile identifier: coap_group_oscore_app (see Section 21.3).

- o REQ13 - Specify policies at the KDC to handle member ids that are not included in 'get_pub_keys': see Section 10.

- o REQ14 - If used, specify the format and content of 'group_policies' and its entries: see Section 6.4; see the three values defined and registered as content of the entry "Sequence Number Synchronization Method" (see Section 21.4).

- o REQ15 - Specify the format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label: see Section 9.

- o REQ16 - Specify how the communication is secured between the Client and KDC: by means of any transport profile of ACE [I-D.ietf-ace-oauth-authz] between Client and Group Manager that complies with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

- o REQ17 - Specify how the nonce N_S is generated, if the token is not being posted (e.g. if it is used directly to validate TLS instead): see Section 6.2.1.

- o REQ18 - Specify if 'mgt_key_material' is used, and if yes specify its format and content: not used in this version of the profile.

- o REQ19 - Define the initial value of the 'num' parameter: The initial value MUST be set to 0 when creating the OSCORE group, e.g. as in [I-D.ietf-ace-oscore-gm-admin].

- o OPT1 (Optional) - Specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: no.

- o OPT2a (Optional) - Specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' is not used: possible early discovery by using the approach based on the CoRE Resource Directory described in [I-D.tiloca-core-oscore-discovery].

- OPT2b (Optional) - Specify additional parameters used in the Token Post exchange: 'ecdh_info', to negotiate the ECDH algorithm, ECDH algorithm parameters, ECDH key parameters and exact encoding of public keys used in the group, in case the joining node supports the pairwise mode of Group OSCORE.

- OPT3 (Optional) - Specify the encoding of 'pub_keys_repos' if the default is not used: no.

- OPT4 (Optional) - Specify policies that instruct clients to retain unsuccessfully decrypted messages and for how long, so that they can be decrypted after getting updated keying material: no.

- OPT5 (Optional) - Specify the behavior of the handler in case of failure to retrieve a public key for the specific node: send a 4.00 Bad Request response to a Joining Request (see Section 6.3).

- OPT6 (Optional) - Specify possible or required payload formats for specific error cases: send a 4.00 Bad Request response to a Joining Request (see Section 6.3).

- OPT7 (Optional) - Specify CBOR values to use for abbreviating identifiers of roles in the group or topic: see Section 4.1.

- OPT8 (Optional) - Specify for the KDC to perform group rekeying (together or instead of renewing individual keying material) when receiving a Key Renewal Request: the Group Manager SHOULD NOT perform a group rekeying, unless already scheduled to occur shortly (see Section 9).

- OPT9 (Optional) - Specify the functionalities implemented at the 'control_path' resource hosted at the Client, including message exchange encoding and other details (see Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm]): see Section 17 for the eviction of a group member; see Section 18 for the group rekeying process.

- OPT10 (Optional) - Specify how the identifier of the sender's public key is included in the group request: no.

Appendix B. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

B.1. Version -08 to -09

- The url-path "ace-group" is used.
- Added overview of admitted methods on the Group Manager resources.
- Added exchange of parameters relevant for the pairwise mode of Group OSCORE.
- The signed value for 'client_cred_verify' includes also the scope.
- Renamed the key material object as Group_OSCORE_Input_Material object.
- Replaced 'clientId' with 'group_SenderId'.
- Added message exchange for Group Names request-response.
- No reassignment of Sender ID and Gid in the same OSCORE group.
- Updates on group rekeying contextual with request of new Sender ID.
- Signature verifiers can also retrieve Group Names and URIs.
- Removed group policy about supporting Group OSCORE in pairwise mode.
- Registration of the resource type rt="core.osc.gm".
- Update list of requirements.
- Clarifications and editorial revision.

B.2. Version -07 to -08

- AIF specific data model to express scope entries.
- A set of roles is checked as valid when processing the Joining Request.
- Updated format of 'get_pub_keys' in the Joining Request.
- Payload format and default values of group policies in the Joining Response.
- Updated payload format of the FETCH request to retrieve public keys.

- Default values for group configuration parameters.
- IANA registrations to support the AIF specific data model.

B.3. Version -06 to -07

- Alignments with draft-ietf-core-oscore-groupcomm.
- New format of 'sign_info', using the COSE capabilities.
- New format of Joining Response parameters, using the COSE capabilities.
- Considerations on group rekeying.
- Editorial revision.

B.4. Version -05 to -06

- Added role of external signature verifier.
- Parameter 'rsnonce' renamed to 'kdcchallenge'.
- Parameter 'kdcchallenge' may be omitted in some cases.
- Clarified difference between group name and OSCORE Gid.
- Removed the role combination ["requester", "monitor"].
- Admit implicit scope and audience in the Authorization Request.
- New format for the 'sign_info' parameter.
- Scope not mandatory to include in the Joining Request.
- Group policy about supporting Group OSCORE in pairwise mode.
- Possible individual rekeying of a single requesting node combined with a group rekeying.
- Security considerations on reusage of signature challenges.
- Addressing optional requirement OPT9 from draft-ietf-ace-key-groupcomm
- Editorial improvements.

B.5. Version -04 to -05

- Nonce N_S also in error responses to the Joining Requests.
- Supporting single Access Token for multiple groups/topics.
- Supporting legal requesters/responders using the 'peer_roles' parameter.
- Registered and used dedicated label for TLS Exporter.
- Added method for uploading a new public key to the Group Manager.
- Added resource and method for retrieving the current group status.
- Fixed inconsistency in retrieving group keying material only.
- Clarified retrieval of keying material for monitor-only members.
- Clarification on incrementing version number when rekeying the group.
- Clarification on what is re-distributed with the group rekeying.
- Security considerations on the size of the nonces used for the signature challenge.
- Added CBOR values to abbreviate role identifiers in the group.

B.6. Version -03 to -04

- New abstract.
- Moved general content to draft-ietf-ace-key-groupcomm
- Terminology: node name; node resource.
- Creation and pointing at node resource.
- Updated Group Manager API (REST methods and offered services).
- Size of challenges 'cnonce' and 'rsnonce'.
- Value of 'rsnonce' for reused or non-traditionally-posted tokens.
- Removed reference to RFC 7390.
- New requirements from draft-ietf-ace-key-groupcomm

- Editorial improvements.

B.7. Version -02 to -03

- New sections, aligned with the interface of ace-key-groupcomm .
- Exchange of information on the countersignature algorithm and related parameters, during the Token POST (Section 4.1).
- Nonce 'rsnonce' from the Group Manager to the Client (Section 4.1).
- Client PoP signature in the Key Distribution Request upon joining (Section 4.2).
- Local actions on the Group Manager, upon a new node's joining (Section 4.2).
- Local actions on the Group Manager, upon a node's leaving (Section 12).
- IANA registration in ACE Groupcomm Parameters Registry.
- More fulfilled profile requirements (Appendix A).

B.8. Version -01 to -02

- Editorial fixes.
- Changed: "listener" to "responder"; "pure listener" to "monitor".
- Changed profile name to "coap_group_oscore_app", to reflect it is an application profile.
- Added the 'type' parameter for all requests to a Join Resource.
- Added parameters to indicate the encoding of public keys.
- Challenge-response for proof-of-possession of signature keys (Section 4).
- Renamed 'key_info' parameter to 'sign_info'; updated its format; extended to include also parameters of the countersignature key (Section 4.1).
- Code 4.00 (Bad request), in responses to joining nodes providing an invalid public key (Section 4.3).

- Clarifications on provisioning and checking of public keys (Sections 4 and 6).

- Extended discussion on group rekeying and possible different approaches (Section 7).

- Extended security considerations: proof-of-possession of signature keys; collision of OSCORE Group Identifiers (Section 8).

- Registered three entries in the IANA Registry "Sequence Number Synchronization Method Registry" (Section 9).

- Registered one public key encoding in the "ACE Public Key Encoding" IANA Registry (Section 9).

B.9. Version -00 to -01

- Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).

- Added negotiation of countersignature algorithm/parameters between Client and Group Manager (Section 4).

- Updated format of the Key Distribution Response as a whole (Section 4.3).

- Added parameter 'cs_params' in the 'key' parameter of the Key Distribution Response (Section 4.3).

- New IANA registrations in the "ACE Authorization Server Request Creation Hints" Registry, "ACE Groupcomm Key" Registry, "OSCORE Security Context Parameters" Registry and "ACE Groupcomm Profile" Registry (Section 9).

Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Carsten Bormann, Martin Gunnarsson, Rikard Hoeglund, Daniel Migault, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; by the H2020 project SIFIS-Home (Grant agreement 952652); and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-164 29 Stockholm
Sweden

Email: marco.tiloca@ri.se

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 21, 2021
December 18, 2020

C. Sengul
Brunel University
A. Kirby
Oxbotica

Message Queuing Telemetry Transport (MQTT)-TLS profile of Authentication and Authorization for Constrained Environments (ACE) Framework
draft-ietf-ace-mqtt-tls-profile-10

Abstract

This document specifies a profile for the ACE (Authentication and Authorization for Constrained Environments) framework to enable authorization in a Message Queuing Telemetry Transport (MQTT)-based publish-subscribe messaging system. Proof-of-possession keys, bound to OAuth2.0 access tokens, are used to authenticate and authorize MQTT Clients. The protocol relies on TLS for confidentiality and MQTT server (broker) authentication.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   3
     1.1.  Requirements Language . . . . . . . . . . . . . . . . . .   4
     1.2.  ACE-Related Terminology . . . . . . . . . . . . . . . . .   4
     1.3.  MQTT-Related Terminology  . . . . . . . . . . . . . . . .   5
   2.  Authorizing Connection Requests . . . . . . . . . . . . . . .   7
     2.1.  Client Token Request to the Authorization Server (AS) . .   8
     2.2.  Client Connection Request to the Broker (C) . . . . . . .   9
       2.2.1.  Client-Server Authentication over TLS and MQTT  . . .   9
       2.2.2.  authz-info: The Authorization Information Topic . . .  10
       2.2.3.  Transporting Access Token Inside the MQTT CONNECT . .  11
       2.2.4.  Authentication Using AUTH Property  . . . . . . . . .  14
         2.2.4.1.  Proof-of-Possession Using a Challenge from the
                   TLS session . . . . . . . . . . . . . . . . . . .  14
         2.2.4.2.  Proof-of-Possession via Broker-generated
                   Challenge/Response  . . . . . . . . . . . . . . .  15
       2.2.5.  Token Validation  . . . . . . . . . . . . . . . . . .  16
       2.2.6.  The Broker's Response to Client Connection Request  .  16
         2.2.6.1.  Unauthorised Request and the Optional
                   Authorisation Server Discovery  . . . . . . . . .  17
         2.2.6.2.  Authorisation Success . . . . . . . . . . . . . .  17
   3.  Authorizing PUBLISH and SUBSCRIBE Messages  . . . . . . . . .  17
     3.1.  PUBLISH Messages from the Publisher Client to the Broker   18
     3.2.  PUBLISH Messages from the Broker to the Subscriber
           Clients . . . . . . . . . . . . . . . . . . . . . . . . .  19
     3.3.  Authorizing SUBSCRIBE Messages  . . . . . . . . . . . . .  19
   4.  Token Expiration, Update and Reauthentication . . . . . . . .  20
   5.  Handling Disconnections and Retained Messages . . . . . . . .  20
   6.  Reduced Protocol Interactions for MQTT v3.1.1 . . . . . . . .  21
     6.1.  Token Transport . . . . . . . . . . . . . . . . . . . . .  21
     6.2.  Handling Authorization Errors . . . . . . . . . . . . . .  22
   7.  IANA Considerations . . . . . . . . . . . . . . . . . . . . .  23
   8.  Security Considerations . . . . . . . . . . . . . . . . . . .  24
   9.  Privacy Considerations  . . . . . . . . . . . . . . . . . . .  25
   10. References  . . . . . . . . . . . . . . . . . . . . . . . . .  25
     10.1.  Normative References . . . . . . . . . . . . . . . . . .  25
     10.2.  Informative References . . . . . . . . . . . . . . . . .  27
   Appendix A.  Checklist for profile requirements . . . . . . . .  28
   Appendix B.  Document Updates . . . . . . . . . . . . . . . . .  29
   Acknowledgements  . . . . . . . . . . . . . . . . . . . . . . .  33
   Authors' Addresses  . . . . . . . . . . . . . . . . . . . . . .  33
```

1. Introduction

This document specifies a profile for the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, Clients and Servers (Brokers) use MQTT to exchange Application Messages. The protocol relies on TLS for communication security between entities. The MQTT protocol interactions are described based on the MQTT v5.0 - the OASIS Standard [MQTT-OASIS-Standard-v5]. Since it is expected that MQTT deployments will continue to support MQTT v3.1.1 clients, this document also describes a reduced set of protocol interactions for MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard]. However, MQTT v5.0 is the RECOMMENDED version as it works more naturally with ACE-style authentication and authorization.

MQTT is a publish-subscribe protocol and after connecting to the MQTT Server (Broker), a Client can publish and subscribe to multiple topics. The Broker, which acts as the Resource Server (RS), is responsible for distributing messages published by the publishers to their subscribers. In the rest of the document the terms "RS", "MQTT Server" and "Broker" are used interchangeably.

Messages are published under a Topic Name, and subscribers subscribe to the Topic Names to receive the corresponding messages. The Broker uses the Topic Name in a published message to determine which subscribers to relay the messages. In this document, topics, more specifically, Topic Names, are treated as resources. The Clients are assumed to have identified the publish/subscribe topics of interest out-of-band (topic discovery is not a feature of the MQTT protocol). A Resource Owner can pre-configure policies at the Authorisation Server (AS) that give Clients publish or subscribe permissions to different topics.

Clients prove their permission to publish and subscribe to topics hosted on an MQTT broker using an access token, bound to a proof-of-possession (PoP) key. This document describes how to authorize the following exchanges between the Clients and the Broker.

o Connection requests from the Clients to the Broker

o Publish requests from the Clients to the Broker, and from the Broker to the Clients

o Subscribe requests from Clients to the Broker

Clients use MQTT PUBLISH message to publish to a topic. This document does not protect the payload of the PUBLISH message from the Broker. Hence, the payload is not signed or encrypted specifically for the subscribers. This functionality MAY be implemented using the proposal outlined in the ACE Pub-Sub Profile [I-D.ietf-ace-pubsub-profile].

To provide communication confidentiality and RS authentication, TLS is used, and TLS 1.3 [RFC8446] is RECOMMENDED. This document makes the same assumptions as Section 4 of the ACE framework [I-D.ietf-ace-oauth-authz] regarding Client and RS registration with the AS and setting up keying material. While the Client-Broker exchanges are only over MQTT, the required Client-AS and RS-AS interactions are described for HTTPS-based communication [RFC7230], using 'application/ace+json' content type, and unless otherwise specified, using JSON encoding. The token MAY be a reference or JSON Web Token (JWT) [RFC7519]. For JWTs, this document follows [RFC7800] for PoP semantics for JWTs. The Client-AS and RS-AS MAY also use protocols other than HTTP, e.g. Constrained Application Protocol (CoAP) [RFC7252] or MQTT. Implementations MAY also use "application/ace+cbor" content type, and CBOR encoding [RFC8949], and CBOR Web Token (CWT) [RFC8392] and associated PoP semantics to reduce the protocol memory and bandwidth requirements. For more information, see Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs) [RFC8747].

1.1. Requirements Language

The keywords "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.

1.2. ACE-Related Terminology

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify" are taken from [RFC4949].

The terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] such as "Client" (C), "Resource Server" (RS) and "Authorization Server" (AS).

The term "resource" is used to refer to an MQTT Topic Name, which is defined in Section 1.3. Hence, the "Resource Owner" is any entity that can authoritatively speak for the topic. This document also defines a Client Authorisation Server, for Clients that are not able to support HTTP.

Client Authorization Server (CAS)

An entity that prepares and endorses authentication and authorization data for a Client, and communicates using HTTPS to the AS.

1.3. MQTT-Related Terminology

The document describes message exchanges as MQTT protocol interactions. The Clients are MQTT Clients, which connect to the Broker to publish and subscribe to Application Messages, labelled with their topics. For additional information, please refer to the MQTT v5.0 - the OASIS Standard [MQTT-OASIS-Standard-v5] or the MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard].

MQTTS
Secured transport profile of MQTT. MQTTS runs over TLS.

Broker
The Server in MQTT. It acts as an intermediary between the Clients that publish Application Messages, and the Clients that made Subscriptions. The Broker acts as the Resource Server for the Clients.

Client
A device or program that uses MQTT.

Session
A stateful interaction between a Client and a Broker. Some Sessions last only as long as the network connection, others can span multiple network connections.

Application Message
The data carried by the MQTT protocol. The data has an associated Quality-of-Service (QoS) level and a Topic Name.

QoS level
The level of assurance for the delivery of an Application Message. The QoS level can be 0-2, where "0" indicates "At most once delivery", "1" "At least once delivery", and "2" "Exactly once delivery".

Property
The last field of the Variable Header is a set of properties for several MQTT control messages (e.g. CONNECT, CONNACK) . A Property consists of an Identifier which defines its usage and data type, followed by a value. The Identifier is encoded as a Variable Byte Integer. For example, "Authentication Data" property with an Identifier 22.

Topic Name
: The label attached to an Application Message, which is matched to a Subscription.

Subscription
: A Subscription comprises a Topic Filter and a maximum QoS. A Subscription is associated with a single session.

Topic Filter
: An expression that indicates interest in one or more Topic Names. Topic Filters may include wildcards.

MQTT sends various control messages across a network connection. The following is not an exhaustive list and the control packets that are not relevant for authorization are not explained. These include, for instance, the PUBREL and PUBCOMP packets used in the 4-step handshake required for QoS level 2.

CONNECT
: Client request to connect to the Broker. This is the first packet sent by a Client.

CONNACK
: The Broker connection acknowledgment. CONNACK packets contain return codes indicating either a success or an error state in response to a Client's CONNECT packet.

AUTH
: Authentication Exchange. An AUTH control packet is sent from the Client to the Broker or from the Broker to the Client as part of an extended authentication exchange. AUTH Properties include Authentication Method and Authentication Data. The Authentication Method is set in the CONNECT packet, and consequent AUTH packets follow the same Authentication Method. The contents of the Authentication Data are defined by the Authentication Method.

PUBLISH
: Publish request sent from a publishing Client to the Broker, or from the Broker to a subscribing Client.

PUBACK
: Response to a PUBLISH request with QoS level 1. A PUBACK can be sent from the Broker to a Client or from a Client to the Broker.

PUBREC

Response to PUBLISH request with QoS level 2. PUBREC can be sent from the Broker to a Client or from a Client to the Broker.

SUBSCRIBE
: Subscribe request sent from a Client.

SUBACK
: Subscribe acknowledgment.

PINGREQ
: A ping request sent from a Client to the Broker. It signals to the Broker that the Client is alive, and is used to confirm that the Broker is also alive. The "Keep Alive" period is set in the CONNECT message.

PINGRESP
: Response sent by the Broker to the Client in response to PINGREQ. It indicates the Broker is alive.

Will
: If the network connection is not closed normally, the Broker sends a last Will message for the Client, if the Client provided one in its CONNECT message. If the Will Flag is set in the CONNECT flags, then the payload of the CONNECT message includes information about the Will. The information consists of the Will Properties, Will Topic, and Will Payload fields.

2. Authorizing Connection Requests

This section specifies how Client connections are authorized by the MQTT Broker. Figure 1 shows the basic protocol flow during connection set-up. The token request and response use the token endpoint at the AS, specified in Section 5.6 of the ACE framework [I-D.ietf-ace-oauth-authz]. Steps (D) and (E) are optional and use the introspection endpoint, specified in Section 5.7 of the ACE framework. The Client and the Broker use HTTPS to communicate to AS via these endpoints. The Client and the Broker use MQTT to communicate between them. The C-AS and Broker-AS communication MAY be implemented using protocols other than HTTPS, e.g. CoAP or MQTT.

If the Client is resource-constrained or does not support HTTPS, a separate Client Authorisation Server may carry out the token request on behalf of the Client, and later, onboard the Client with the token. The interactions between a Client and its Client Authorization Server for token onboarding, and support for MQTTS-based token requests at the AS are out of scope of this document.

```
                            +---------------------+
                            | Client              |
                            |                     |
     +---(A) Token request--| Client -            |
     |                      | Authorization       |
     |   +-(B) Access token-> Server Interface    |
     |   |                  |       (HTTPS)       |
     |   |                  |_____________________|
     |   |                  |                     |
  +--v-------------+        |  Pub/Sub Interface  |
  | Authorization  |        |      (MQTTS)        |
  | Server         |        +-----------^---------+
  |________________|            |       |
     |    ^                (C)Connection  (F)Connection
     |    |                  request +    response
     |    |                  access token |
     |    |                     |        |
     |    |                 +---v--------------+
     |    |                 |   Broker (MQTTS)  |
     |    |                 |__________________|
     |    +(D)Introspection-|                  |
     |   request (optional) | RS-AS interface  |
     |                      |      (HTTPS)     |
     +-(E)Introspection---->|__________________|
       response (optional)
```

Figure 1: Connection set-up

2.1. Client Token Request to the Authorization Server (AS)

The first step in the protocol flow (Figure 1 (A)) is the token acquisition by the Client from the AS. The Client and the AS MUST perform mutual authentication. The Client requests an access token from the AS as described in Section 5.6.1 of the ACE framework [I-D.ietf-ace-oauth-authz]. The media format is 'application/ace+json'. The AS uses JSON in the payload of its responses to the Client and the RS.

If the AS successfully verifies the access token request and authorizes the Client for the indicated audience (i.e. RS) and scopes (i.e. publish/subscribe permissions over topics as described in Section 3), the AS issues an access token (Figure 1 (B)). The response includes the parameters described in Section 5.6.2 of the ACE framework [I-D.ietf-ace-oauth-authz], and specifically, the "ace_profile" parameter is set to "mqtt_tls". The returned token is a Proof-of-Possession (PoP) token by default. This document follows [RFC7800] for PoP semantics for JWTs. The PoP token includes a 'cnf' parameter with a symmetric or asymmetric PoP key. Note that the 'cnf' parameter in the web tokens are to be consumed by the RS and not the Client. For the asymmetric case, the PoP token may include the 'rs_cnf' parameter containing the information about the public key to be used by the RS to authenticate as described in [I-D.ietf-ace-oauth-params].

The AS returns error responses for JSON-based interactions following Section 5.2 of [RFC6749]. When CBOR is used, the interactions MUST implement Section 5.6.3 of the ACE framework [I-D.ietf-ace-oauth-authz].

2.2. Client Connection Request to the Broker (C)

2.2.1. Client-Server Authentication over TLS and MQTT

The Client and the Broker MUST perform mutual authentication. The Client MUST authenticate to the Broker either over MQTT or TLS. For MQTT, the options are "None" and "ace". For TLS, the options are "Anon" for an anonymous client, and "Known(RPK/PSK)" for Raw Public Keys (RPK) [RFC7250] and Pre-Shared Keys (PSK), respectively. Combined, client authentication has the following options:

o "TLS:Anon-MQTT:None": This option is used only for the topics that do not require authorization, including the "authz-info" topic. Publishing to the "authz-info" topic is described in Section 2.2.2.

o "TLS:Anon-MQTT:ace": The token is transported inside the CONNECT message, and MUST be validated using one of the methods described in Section 2.2.2. This option also supports a tokenless connection request for AS discovery.

o "TLS:Known(RPK/PSK)-MQTT:none": For the RPK, the token MUST have been published to the "authz-info" topic. For the PSK, the token MAY be, alternatively, provided as an "identity" in the "identities" field in the client's "pre_shared_key" extension in TLS 1.3. The TLS session set-up is as described in DTLS profile for ACE [I-D.ietf-ace-dtls-authorize].

o "TLS:Known(RPK/PSK)-MQTT:ace": This option SHOULD NOT be chosen as the token transported in the CONNECT overwrites any permissions passed during the TLS authentication.

It is RECOMMENDED that the Client implements "TLS:Anon-MQTT:ace" as a first choice when working with protected topics. However, depending on the Client capability, Client MAY implement "TLS:Known(RPK/PSK)-MQTT:none", and consequently "TLS:Anon-MQTT:None" to submit its token to "authz-info".

The Broker MUST support "TLS:Anon-MQTT:ace". To support Clients with different capabilities, the Broker MAY provide multiple client authentication options, e.g. support "TLS:Known(RPK)-MQTT:none" and "TLS:Anon-MQTT:None", to enable RPK-based client authentication, but fall back to "TLS:Anon-MQTT:ace" if the Client does not send a client certificate (i.e. it sends an empty Certificate message) during the TLS handshake.

The Broker MUST be authenticated during the TLS handshake. If the Client authentication uses TLS:Known(RPK/PSK), then the Broker is authenticated using the respective method. Otherwise, to authenticate the Broker, the client MUST validate a public key from a X.509 certificate or an RPK from the Broker against the 'rs_cnf' parameter in the token response. The AS MAY include the thumbprint of the RS's X.509 certificate in the 'rs_cnf' (thumbprint as defined in [I-D.ietf-cose-x509]). In this case, the client MUST validate the RS certificate against this thumbprint.

2.2.2. authz-info: The Authorization Information Topic

In the cases when the Client MUST transport the token to the Broker first, the Client connects to the Broker to publish its token to the "authz-info" topic. The "authz-info" topic MUST be publish-only (i.e. the Clients are not allowed to subscribe to it). "authz-info" is not protected, and hence, the Client uses the "TLS:Anon-MQTT:None" option over a TLS connection. After publishing the token, the Client disconnects from the Broker and is expected to reconnect using client authentication over TLS (i.e. TLS:Known(RPK/PSK)-MQTT:none).

The Broker stores and indexes all tokens received to the "authz-info" topic in its key store (similar to DTLS profile for ACE [I-D.ietf-ace-dtls-authorize]). This profile follows the recommendation of Section 5.8.1 of the ACE framework [I-D.ietf-ace-oauth-authz], and expects that the Broker stores only one token per proof-of-possession key, and any other token linked to the same key overwrites an existing token.

The Broker MUST verify the validity of the token (i.e. through local validation or introspection, if the token is a reference) as described in Section 2.2.5. If the token is not valid, the Broker MUST discard the token. Depending on the QoS level of the PUBLISH message, the Broker returns the error response as a PUBACK or a DISCONNECT message as explained below.

If the QoS level is equal to 0, and the token is invalid or the claims cannot be obtained in the case of an introspected token, the Broker MUST send a DISCONNECT message with the reason code '0x87 (Not authorized)'. If the PUBLISH payload does not parse to a token, the

RS MUST send a DISCONNECT with the reason code '0x99 (Payload format invalid)'.

If the QoS level of the PUBLISH message is greater than or equal to 1, the Broker MUST return 'Not authorized' in PUBACK. If the PUBLISH payload does not parse to a token, the PUBACK reason code is '0x99 (Payload format invalid)'.

It must be noted that when the RS sends the 'Not authorized' response, this corresponds to the token being invalid, and not that the actual PUBLISH message was not authorized. Given that the "authz-info" is a public topic, this response is not expected to cause confusion.

2.2.3. Transporting Access Token Inside the MQTT CONNECT

This section describes how the Client transports the token to the Broker (RS) inside the CONNECT message. If this method is used, the Client TLS connection is expected to be anonymous, and the Broker is authenticated during the TLS connection set-up. The approach described in this section is similar to an earlier proposal by Fremantle et al [fremantle14].

After sending the CONNECT, the client MUST wait to receive the CONNACK from the Broker. The only messages it is allowed to send are DISCONNECT or AUTH that is in response to the Broker AUTH. Similarly, the Broker MUST NOT process any packets before it has sent a CONNACK. The only exceptions are DISCONNECT or an AUTH response from the Client.

Figure 2 shows the structure of the MQTT CONNECT message used in MQTT v5.0. A CONNECT message is composed of a fixed header, a variable header and a payload. The fixed header contains the Control Packet Type (CPT), Reserved, and Remaining Length fields. The Variable Header contains the Protocol Name, Protocol Level, Connect Flags, Keep Alive, and Properties fields. The Connect Flags in the variable header specify the properties of the MQTT session. It also indicates the presence or absence of some fields in the Payload. The payload contains one or more encoded fields, namely a unique Client identifier for the Client, a Will Topic, Will Payload, User Name and Password. All but the Client identifier can be omitted depending on the flags in the Variable Header.

```
0             8             16             24            32
+-------------------------------------------------------+
|CPT=1 | Rsvd.|Remaining len.| Protocol name len. = 4  |
+-------------------------------------------------------+
|                     'M' 'Q' 'T' 'T'                   |
+-------------------------------------------------------+
| Proto.level=5|Connect flags|          Keep alive      |
+-------------------------------------------------------+
|                 Property length                       |
|          Auth. Method (0x15) | 'ace'                  |
|          Auth. Data (0x16)   | token or               |
|                                token + PoP data       |
+-------------------------------------------------------+
|                      Payload                          |
+-------------------------------------------------------+
```

Figure 2: MQTT v5 CONNECT control message with ACE authentication.
(CPT=Control Packet Type)

The CONNECT message flags are Username, Password, Will retain, Will QoS, Will Flag, Clean Start, and Reserved. Figure 3 shows how the flags MUST be set to use AUTH packets for authentication and authorisation, i.e. the username and password flags MUST be set to 0. An MQTT v5.0 RS MAY also support token transport using Username and Password to provide a security option for MQTT v3.1.1 clients, as described in Section 6.

User name Flag	Pass. Flag	Will retain	Will QoS	Will Flag	Clean Start	Rsvd.
0	0	X	X X	X	X	0

Figure 3: CONNECT flags for AUTH

The Will Flag indicates that a Will message needs to be sent if the network connection is not closed normally. The situations in which the Will message is published include disconnections due to I/O or network failures, and the server closing the network connection due to a protocol error. The Client MAY set the Will Flag as desired (marked as 'X' in Figure 3). If the Will Flag is set to 1 and the Broker accepts the connection request, the Broker stores the Will message and publish it when the network connection is closed according to Will QoS and Will retain parameters and MQTT Will management rules. To avoid publishing Will Messages in the case of temporary network disconnections, the Client specifies a Will Delay

Interval in the Will Properties. Section 5 explains how the Broker deals with the retained messages in further detail.

In MQTT v5.0, the Client signals a clean session (i.e. the session does not continue an existing session), by setting the Clean Start Flag to 1 and, the Session Expiry Interval to 0 in the CONNECT message. In this profile, the Broker SHOULD always start with a clean session regardless of how these parameters are set. Starting a clean session helps the Broker avoid keeping unnecessary session state for unauthorised clients. If the Broker starts a clean session, the Broker MUST set the Session Present flag to 0 in the CONNACK packet to signal this to the Client.

The Broker MAY support session continuation e.g., if the Broker requires it for QoS reasons. With session continuation, the Broker maintains and uses client state from the existing session. The session state kept at the server MAY include token and its introspection result (for reference tokens) in addition to the MQTT session state. The MQTT session state is identified by the Client identifier and includes state on client subscriptions; messages with QoS levels 1 and 2, and which have not been completely acknowledged or pending transmission to the Client; and if the Session is currently not connected, the time at which the Session will end and Session State will be discarded.

When reconnecting to a Broker that supports session continuation, the Client MUST still provide a token, in addition to using the same Client identifier, setting the Clean Start to 0 and supplying a Session Expiry interval in the CONNECT message. The Broker MUST perform proof-of-possession validation on the provided token. If the token matches the stored state, the Broker MAY skip introspecting a token by reference, and use the stored introspection result. The Broker MUST also verify the Client is authorized to receive or send packets that are pending transmission. When a Client connects with a long Session Expiry Interval, the Broker may need to maintain Client's MQTT session state after it disconnects for an extended period. Brokers SHOULD implement administrative policies to limit misuse.

Note that, according to the MQTT standard, the Broker uses the Client identifier to identify the session state. In the case of a Client identifier collision, a client may take over another client's session. Given that clients provide a token at each connection, clients will only send or receive messages to their authorized topics. Therefore, while this issue is not expected to affect security, it may affect QoS (i.e. PUBLISH or QoS messages saved for Client A may be delivered to a Client B). In addition, if this Client identifier represents a Client already connected to the

Broker, the Broker sends a DISCONNECT packet to the existing Client with Reason Code of '0x8E (Session taken over)', and closes the connection to the client.

2.2.4. Authentication Using AUTH Property

To use AUTH, the Client MUST set the Authentication Method as a property of a CONNECT packet by using the property identifier 21 (0x15). This is followed by a UTF-8 Encoded String containing the name of the Authentication Method, which MUST be set to 'ace'. If the RS does not support this profile, it sends a CONNACK with a Reason Code of '0x8C (Bad authentication method)'.

The Authentication Method is followed by the Authentication Data, which has a property identifier 22 (0x16) and is binary data. The binary data in MQTT is represented by a two-byte integer length, which indicates the number of data bytes, followed by that number of bytes. Based on the Authentication Data, RS MUST support both options below:

- o Proof-of-Possession using a challenge from the TLS session

- o Proof-of-Possession via Broker generated challenge/response

2.2.4.1. Proof-of-Possession Using a Challenge from the TLS session

```
+-----------------------------------------------------------------+
|Authentication|Token Length|Token    |MAC or Signature           |
|Data Length   |            |         |(over TLS exporter content) |
+-----------------------------------------------------------------+
```

Figure 4: Authentication Data for PoP based on TLS exporter content

For this option, the Authentication Data MUST contain the two-byte integer token length, the token, and the keyed message digest (MAC) or the Client signature (as shown in Figure 4). The Proof-of-Possession key in the token is used to calculate the keyed message digest (MAC) or the Client signature based on the content obtained from the TLS exporter ([RFC5705] for TLS 1.2 and for TLS 1.3, Section 7.5 of [RFC8446]). This content is exported from the TLS session using the exporter label 'EXPORTER-ACE-MQTT-Sign-Challenge', an empty context, and length of 32 bytes. The token is also validated as described in Section 2.2.5 and the server responds with a CONNACK with the appropriate response code. The client cannot reauthenticate using this method during the same session (see Section 4).

2.2.4.2. Proof-of-Possession via Broker-generated Challenge/Response

```
+------------------------------------+
|Authentication|Token Length|Token   |
|Data Length   |            |        |
+------------------------------------+
```

Figure 5: Authentication Data to Initiate PoP based on Challenge/ Response

```
+------------------------------+
|Authentication|Nonce (8 bytes)|
|Data Length   |               |
+------------------------------+
```

Figure 6: Authentication Data for Broker Challenge

For this option, the RS follows a Broker-generated challenge/response protocol. If the Authentication Data contains only the two-byte integer token length and the token (as shown in Figure 5), the RS MUST respond with an AUTH packet, with the Authenticate Reason Code set to "0x18 (Continue Authentication)". This packet includes the Authentication Method, which MUST be set to "ace" and Authentication Data. The Authentication Data MUST NOT be empty and contains an 8-byte nonce as a challenge for the Client (Figure 6).

```
+------------------------------------------------------------------+
|Authentication|Client Nonce    |Client|MAC or Signature             |
|Data Length   |Length          |nonce |(over RS nonce+Client nonce)|
+------------------------------------------------------------------+
```

Figure 7: Authentication Data for Client Challenge Response

The Client responds to this with an AUTH packet with a reason code "0x18 (Continue Authentication)". Similarly, the Client packet sets the Authentication Method to "ace". The Authentication Data in the Client’s response is formatted as shown in Figure 7 and includes the client nonce length, the client nonce, and the signature or MAC computed over the RS nonce concatenated with the client nonce using PoP key in the token.

Next, the token is validated as described in Section 2.2.5. The success case is illustrated in Figure 8. The client MAY also re-authenticate using this challenge-response flow, as described in Section 4.

```
                                Resource
                    Client      Server
                     |             |
                     |<===========>| TLS connection set-up
                     |             |
                     |             |
                     +------------>| CONNECT with Authentication Data
                     |             | contains only token
                     |             |
                     <-------------+ AUTH '0x18 (Cont. Authentication)'
                     |             | 8-byte nonce as RS challenge
                     |             |
                     |------------>| AUTH '0x18 (Cont. Authentication)'
                     |             | 8-byte client nonce + signature/MAC
                     |             |
                     |             |---+ Token validation
                     |             |   | (may involve introspection)
                     |             |<--+
                     |             |
                     |<------------+ CONNACK '0x00 (Success)'
```

Figure 8: PoP Challenge/Response Flow - Success

2.2.5. Token Validation

The RS MUST verify the validity of the token either locally (e.g. in the case of a self-contained token) or the RS MAY send an introspection request to the AS. The RS MUST verify the claims according to the rules set in the Section 5.8.1.1 of the ACE framework [I-D.ietf-ace-oauth-authz].

To authenticate the Client, the RS validates the signature or the MAC, depending on how the PoP protocol is implemented. HS256 (HMAC-SHA-256) [RFC6234] and Ed25519 [RFC8032] are mandatory to implement depending on the choice of symmetric or asymmetric validation. Validation of the signature or MAC MUST fail if the signature algorithm is set to "none", when the key used for the signature algorithm cannot be determined, or the computed and received signature/MAC do not match.

2.2.6. The Broker's Response to Client Connection Request

Based on the validation result (obtained either via local inspection or using the /introspection interface of the AS), the Broker MUST send a CONNACK message to the Client.

2.2.6.1. Unauthorised Request and the Optional Authorisation Server Discovery

If the Client does not provide a valid token or omits the Authentication Data field, or the token or Authentication data are malformed, authentication fails. The Broker responds with the CONNACK reason code "0x87 (Not Authorized)"

The Broker MAY also trigger AS discovery, and include a User Property (identified by 38 (0x26)) in the CONNACK for the AS Request Creation Hints. The User Property is a UTF-8 string pair, composed of a name and a value. The name of the User Property MUST be set to "ace_as_hint". The value of the user property is a UTF-8 encoded JSON string containing the mandatory "AS" parameter, and the optional parameters "audience", "kid", "cnonce", and "scope" as defined in Section 5.1.2 of the ACE framework [I-D.ietf-ace-oauth-authz].

2.2.6.2. Authorisation Success

On success, the reason code of the CONNACK is "0x00 (Success)". The AS informs the client that selected profile is "mqtt_tls" using the "ace_profile" parameter in the token response. If the Broker starts a new session, it MUST also set Session Present to 0 in the CONNACK packet to signal a clean session to the Client. Otherwise, it MUST set Session Present to 1.

If the Broker accepts the connection, it MUST store the token until the end of the connection. On Client or Broker disconnection, the Client is expected to transport a token again on the next connection attempt.

If the token is not self-contained and the Broker uses token introspection, it MAY cache the validation result to authorize the subsequent PUBLISH and SUBSCRIBE messages. PUBLISH and SUBSCRIBE messages, which are sent after a connection set-up, do not contain access tokens. If the introspection result is not cached, then the RS needs to introspect the saved token for each request. The Broker SHOULD also use a cache time out to introspect tokens regularly.

3. Authorizing PUBLISH and SUBSCRIBE Messages

To authorize a Client’s PUBLISH and SUBSCRIBE messages, the Broker uses the scope field in the token (or in the introspection result). The scope field contains the publish and subscribe permissions for the Client. The scope is a JSON array, each item following the Authorization Information Format (AIF) for ACE [I-D.ietf-ace-aif]. Using the Concise Data Definition Language (CDDL) [RFC8610], the specific data model for MQTT is:

```
 AIF-MQTT = AIF-Generic<topic_filter, permissions>
 AIF-Generic<topic_filter, permissions> = [*[topic_filter, permissions]]
 topic_filter = tstr
 permissions = [+permission]
 permission = "pub"/"sub"
```

Figure 9: AIF-MQTT data model

Topic filters are implemented according to Section 4.7 of MQTT v5.0 - the OASIS Standard [MQTT-OASIS-Standard-v5] and includes special wildcard characters. The multi-level wildcard, '#', matches any number of levels within a topic, and the single-level wildcard, '+', matches one topic level.

If the scope is empty i.e., the JSON array is empty, the RS records no permissions for the client for any topic. In this case, the client is not able to publish or subscribe to any protected topics.

An example scope may contain:

```
 [["topic1", ["pub","sub"]], ["topic2/#",["pub"]], ["+/topic3",["sub"]]]
```

Figure 10: Example scope

This access token gives publish ("pub") and subscribe ("sub") permissions to the "topic1", publish permission to all the subtopics of "topic2", and subscribe permission to all "topic3", skipping one level.

If the Will Flag is set, then the Broker MUST check that the token allows the publication of the Will message (i.e. the Will Topic filter is in the scope array).

3.1. PUBLISH Messages from the Publisher Client to the Broker

On receiving the PUBLISH message, the Broker MUST use the type of message (i.e. PUBLISH) and the Topic name in the message header to match against the scope array items in the cached token or its introspection result. Following the example in the previous section, a client sending a PUBLISH message to 'topic2/a' would be allowed, as the scope array includes the '["topic2/#",["pub"]]'.

If the Client is allowed to publish to the topic, the Broker publishes the message to all valid subscribers of the topic. In the case of an authorization failure, the Broker MUST return an error, if the Client has set the QoS level of the PUBLISH message to greater than or equal to 1. Depending on the QoS level, the Broker responds with either a PUBACK or PUBREC packet with reason code '0x87 (Not authorized)’. On receiving an acknowledgement with ’0x87 (Not authorized)’, the Client MAY reauthenticate by providing a new token as described in Section 4.

For QoS level 0, the Broker sends a DISCONNECT with reason code "0x87 (Not authorized)" and closes the network connection. Note that the server-side DISCONNECT is a new feature of MQTT v5.0 (in MQTT v3.1.1, the server needs to drop the connection).

3.2. PUBLISH Messages from the Broker to the Subscriber Clients

To forward PUBLISH messages to the subscribing Clients, the Broker identifies all the subscribers that have valid matching topic subscriptions (i.e. the tokens are valid, and token scopes allow a subscription to the particular topic). The Broker sends a PUBLISH message with the Topic name to all the valid subscribers.

The Broker MUST NOT forward messages to the unauthorized subscribers. There is no way to inform the Clients with invalid tokens that an authorization error has occurred other than sending a DISCONNECT message. The Broker SHOULD send a DISCONNECT message with the reason code ’0x87 (Not authorized)’.

3.3. Authorizing SUBSCRIBE Messages

In MQTT, a SUBSCRIBE message is sent from a Client to the Broker to create one or more subscriptions to one or more topics. The SUBSCRIBE message may contain multiple Topic Filters. The Topic Filters may include wildcard characters.

On receiving the SUBSCRIBE message, the Broker MUST use the type of message (i.e. SUBSCRIBE) and the Topic Filter in the message header to match against the scope field of the stored token or introspection result. The Topic Filters MUST be equal or a subset of at least one of the ’topic_filter’ fields in the scope array found in the Client’s token.

As a response to the SUBSCRIBE message, the Broker issues a SUBACK message. For each Topic Filter, the SUBACK packet includes a return code matching the QoS level for the corresponding Topic Filter. In the case of failure, the return code is 0x87, indicating that the Client is ’Not authorized’. A reason code is returned for each Topic Filter. Therefore, the Client may receive success codes for a subset of its Topic Filters while being unauthorized for the rest.

4. Token Expiration, Update and Reauthentication

The Broker MUST check for token expiration whenever a CONNECT, PUBLISH or SUBSCRIBE message is received or sent. The Broker SHOULD check for token expiration on receiving a PINGREQUEST message. The Broker MAY also check for token expiration periodically, e.g. every hour. This may allow for early detection of a token expiry.

The token expiration is checked by checking the 'exp' claim of a JWT or introspection response, or via performing an introspection request with the AS as described in Section 5.7 of the ACE framework [I-D.ietf-ace-oauth-authz]. Token expirations may trigger the RS to send PUBACK, SUBACK and DISCONNECT messages with return code set to "Not authorized". After sending a DISCONNECT message, the network connection is closed, and no more messages can be sent.

If the Client used the challenge-respose PoP as defined in Section 2.2.4.2, the Client MAY reauthenticate as a response to the PUBACK and SUBACK that signal loss of authorization. The Clients MAY also proactively update their tokens, i.e. before they receive a message with a "Not authorized" return code. To start reauthentication, the Client MUST send an AUTH packet with the reason code "0x19 (Re-authentication)". The Client MUST set the Authentication Method as "ace" and transport the new token in the Authentication Data. The Broker accepts reauthentication requests if the Client has already submitted a token (may be expired) and validated via the challenge-response PoP. Otherwise, the Broker MUST deny the request. If the reauthentication fails, the Broker MUST send a DISCONNECT with the reason code "0x87 (Not Authorized)".

5. Handling Disconnections and Retained Messages

In the case of a Client DISCONNECT, the Broker deletes all the session state but MUST keep the retained messages. By setting a RETAIN flag in a PUBLISH message, the publisher indicates to the Broker to store the most recent message for the associated topic. Hence, the new subscribers can receive the last sent message from the publisher for that particular topic without waiting for the next PUBLISH message. The Broker MUST continue publishing the retained messages as long as the associated tokens are valid.

In case of disconnections due to network errors or server disconnection due to a protocol error (which includes authorization errors), the Will message is sent if the Client supplied a Will in the CONNECT message. The Client's token scope array MUST include the Will Topic. The Will message MUST be published to the Will Topic regardless of whether the corresponding token has expired. In the case of a server-side DISCONNECT, the server returns the '0x87 Not Authorized' return code to the Client.

6. Reduced Protocol Interactions for MQTT v3.1.1

This section describes a reduced set of protocol interactions for the MQTT v3.1.1 Clients. An MQTT v5.0 Broker MAY implement these interactions for the MQTT v3.1.1 clients; MQTT v5.0 clients are NOT RECOMMENDED to use the flows described in this section. Brokers that do not support MQTT v3.1.1 clients return a CONNACK packet with Reason Code '0x84 (Unsupported Protocol Version)' in response to the connection requests.

6.1. Token Transport

As in MQTT v5.0, the token MAY either be transported before by publishing to the "authz-info" topic, or inside the CONNECT message.

In MQTT v3.1.1, after the Client published to the "authz-info" topic, the Broker cannot communicate the result of the token validation as PUBACK reason codes or server-side DISCONNECT messages are not supported. In any case, an invalid token would fail the subsequent TLS handshake, which can prompt the Client to obtain a valid token.

To transport the token to the Broker inside the CONNECT message, the Client uses the username and password fields. Figure 11 shows the structure of the MQTT CONNECT message.

```
0            8             16             24             32
+-------------------------------------------------------+
|CPT=1 | Rsvd.|Remaining len.| Protocol name len. = 4  |
+-------------------------------------------------------+
|                      'M' 'Q' 'T' 'T'                  |
+-------------------------------------------------------+
| Proto.level=4|Connect flags|          Keep alive     |
+-------------------------------------------------------+
| Payload                                               |
|     Client Identifier                                 |
|     Username as access token (UTF-8)                  |
|     Password length (2 Bytes)                         |
|     Password data as signature/MAC (binary)           |
+-------------------------------------------------------+
```

Figure 11: MQTT CONNECT control message. (CPT=Control Packet Type, Rsvd=Reserved, len.=length, Proto.=Protocol)

Figure 12 shows how the MQTT connect flags MUST be set to initiate a connection with the Broker.

User name flag	Pass. flag	Will retain	Will QoS	Will Flag	Clean	Rsvd.
1	1	X	X X	X	X	0

Figure 12: MQTT CONNECT flags. (Rsvd=Reserved)

The Broker SHOULD NOT accept session continuation. To this end, the Broker ignores how the Clean Session Flag is set, and on connection success, the Broker MUST set the Session Present flag to 0 in the CONNACK packet to indicate a clean session to the Client. If the Broker wishes to support session continuation, it MUST still perform proof-of-possession validation on the provided Client token. MQTT v3.1.1 does not use a Session Expiry Interval, and the Client expects that the Broker maintains the session state after it disconnects. However, stored Session state can be discarded as a result of administrator policies, and Brokers SHOULD implement the necessary policies to limit misuse.

The Client MAY set the Will Flag as desired (marked as 'X' in Figure 12). Username and Password flags MUST be set to 1 to ensure that the Payload of the CONNECT message includes both Username and Password fields.

The CONNECT in MQTT v3.1.1 does not have a field to indicate the authentication method. To signal that the Username field contains an ACE token, this field MUST be prefixed with 'ace' keyword, which is followed by the access token. The Password field MUST be set to the keyed message digest (MAC) or signature associated with the access token for proof-of-possession. The Client MUST apply the PoP key on the challenge derived from the TLS session as described in Section 2.2.4.1.

In MQTT v3.1.1, the MQTT Username is a UTF-8 encoded string (i.e. is prefixed by a 2-byte length field followed by UTF-8 encoded character data) and may be up to 65535 bytes. Therefore, an access token that is not a valid UTF-8 MUST be Base64 [RFC4648] encoded. (The MQTT Password allows binary data up to 65535 bytes.)

6.2. Handling Authorization Errors

Handling errors are more primitive in MQTT v3.1.1 due to not having appropriate error fields, error codes, and server-side DISCONNECTs. Therefore, the broker will disconnect on almost any error and may not keep session state, necessitating clients to make a greater effort to ensure that tokens remain valid and not attempt to publish to topics that they do not have permissions for. The following lists how the broker responds to specific errors.

- o CONNECT without a token: It is not possible to support AS discovery via sending a tokenless CONNECT message to the Broker. This is because a CONNACK packet in MQTT v3.1.1 does not include a means to provide additional information to the Client. Therefore, AS discovery needs to take place out-of-band. The tokenless CONNECT attempt MUST fail.

- o Client-RS PUBLISH authorization failure: In the case of a failure, it is not possible to return an error in MQTT v3.1.1. Acknowledgement messages only indicate success. In the case of an authorization error, the Broker SHOULD disconnect the Client. Otherwise, it MUST ignore the PUBLISH message. Also, as DISCONNECT messages are only sent from a Client to the Broker, the server disconnection needs to take place below the application layer.

- o SUBSCRIBE authorization failure: In the SUBACK packet, the return code is 0x80 indicating 'Failure' for the unauthorized topic(s). Note that, in both MQTT versions, a reason code is returned for each Topic Filter.

- o RS-Client PUBLISH authorization failure: When RS is forwarding PUBLISH messages to the subscribed Clients, it may discover that some of the subscribers are no more authorized due to expired tokens. These token expirations SHOULD lead to disconnecting the Client rather than silently dropping messages.

7. IANA Considerations

This document registers 'EXPORTER-ACE-MQTT-Sign-Challenge' (introduced in Section 2.2.4.1 in this document) in the TLS Exporter Label Registry [RFC8447].

In addition, the following registrations are done for the ACE OAuth Profile Registry following the procedure specified in [I-D.ietf-ace-oauth-authz].

Note to the RFC editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification and delete this paragraph.

Name: mqtt_tls

Description: Profile for delegating Client authentication and authorization using MQTT as the application protocol and TLS For transport layer security.

CBOR Value:

Reference: [RFC-XXXX]

8. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. Therefore, the security considerations outlined in [I-D.ietf-ace-oauth-authz] apply to this work.

In addition, the security considerations outlined in MQTT v5.0 - the OASIS Standard [MQTT-OASIS-Standard-v5] and MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard] apply. Mainly, this document provides an authorization solution for MQTT, the responsibility of which is left to the specific implementation in the MQTT standards. In the following, we comment on a few relevant issues based on the current MQTT specifications.

After the RS validates an access token and accepts a connection from a client, it caches the token to authorize a Client’s publish and subscribe requests in an ongoing session. RS does not cache any invalid tokens. If a client’s permissions get revoked but the access token has not expired, the RS may still grant publish/subscribe to revoked topics. If the RS caches the token introspection responses, then the RS SHOULD use a reasonable cache timeout to introspect tokens regularly. When permissions change dynamically, it is expected that AS also follows a reasonable expiration strategy for the access tokens.

The RS may monitor Client behaviour to detect potential security problems, especially those affecting availability. These include repeated token transfer attempts to the public "authz-info" topic, repeated connection attempts, abnormal terminations, and Clients that connect but do not send any data. If the RS supports the public "authz-info" topic, described in Section 2.2.2, then this may be vulnerable to a DDoS attack, where many Clients use the "authz-info" public topic to transport fictitious tokens, which RS may need to store indefinitely.

For MQTT v5.0, when a Client connects with a long Session Expiry Interval, the RS may need to maintain Client’s MQTT session state after it disconnects for an extended period. For MQTT v3.1.1, the session state may need to be stored indefinitely, as it does not have a Session Expiry Interval feature. The RS SHOULD implement administrative policies to limit misuse of the session continuation by the Client.

9. Privacy Considerations

The privacy considerations outlined in [I-D.ietf-ace-oauth-authz] apply to this work.

In MQTT, the RS is a central trusted party and may forward potentially sensitive information between Clients. This document does not protect the contents of the PUBLISH message from the Broker, and hence, the content of the PUBLISH message is not signed or encrypted separately for the subscribers. This functionality may be implemented using the proposal outlined in the ACE Pub-Sub Profile [I-D.ietf-ace-pubsub-profile]. However, this solution would still not provide privacy for other properties of the message such as Topic Name.

10. References

10.1. Normative References

[I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-ietf-ace-aif-00 (work in progress), July 2020.

[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-36 (work in progress), November 2020.

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-13 (work in progress), April 2020.

[I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", draft-ietf-cose-x509-08 (work in progress), December 2020.

[MQTT-OASIS-Standard]
Banks, A., Ed. and R. Gupta, Ed., "OASIS Standard MQTT Version 3.1.1 Plus Errata 01", 2015, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

[MQTT-OASIS-Standard-v5]
Banks, A., Ed., Briggs, E., Ed., Borgendale, K., Ed., and R. Gupta, Ed., "OASIS Standard MQTT Version 5.0", 2017, <http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[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>.

[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>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[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>.

[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>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

[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>.

[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>.

10.2. Informative References

[fremantle14]
Fremantle, P., Aziz, B., Kopecky, J., and P. Scott, "Federated Identity and Access Management for the Internet of Things", research International Workshop on Secure Internet of Things, September 2014, <http://dx.doi.org/10.1109/SIoT.2014.8>.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-14 (work in progress), October 2020.

[I-D.ietf-ace-pubsub-profile]
Palombini, F., "Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-pubsub-profile-01 (work in progress), July 2020.

[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>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[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>.

Appendix A. Checklist for profile requirements

- o AS discovery: AS discovery is possible with the MQTT v5.0 described in Section 2.2.

- o The communication protocol between the Client and RS: MQTT

- o The security protocol between the Client and RS: TLS

- o Client and RS mutual authentication: Several options are possible and described in Section 2.2.1.

- o Content format: For the HTTPS interactions with AS, "application/ace+json".

- o PoP protocols: Either symmetric or asymmetric keys can be supported.

- o Unique profile identifier: mqtt_tls

- o Token introspection: RS uses HTTPS /introspect interface of AS.

- o Token request: Client or its Client AS uses HTTPS /token interface of AS.

- /authz-info endpoint: It MAY be supported using the method described in Section 2.2.2, but is not protected.

- Token transport: Via "authz-info" topic, or in MQTT CONNECT message for both versions of MQTT. AUTH extensions also used for authentication and re-authentication for MQTT v5.0 as described in Section 2.2 and in Section 4.

Appendix B. Document Updates

Version 07 to 08:

- Fixed several nits, typos based on WG reviews.

- Added missing references.

- Added the definition for Property defined by MQTT, and Client Authorisation Server.

- Added artwork to show Authorisation Data format for various PoP-related message exchange.

- Removed all MQTT-related must/should/may.

- Made AS discovery optional.

- Clarified what the client and server must implement for client authentication; cleaned up TLS 1.3 related language.

Version 06 to 07:

- Corrected the title.

- In Section 2.2.3, added the constraint on which packets the Client can send, and the server can process after CONNECT before CONNACK.

- In Section 2.2.3, clarified that session state is identified by Client Identifier, and listed its content.

- In Section 2.2.3, clarified the issue of Client Identifier collision, when the broker supports session continuation.

- Corrected the buggy scope example in Section 3.1.

Version 05 to 06:

- Replace the originally proposed scope format with AIF model. Defined the AIF-MQTT, gave an example with a JSON array. Added a normative reference to the AIF draft.

- Clarified client connection after submitting token via "authz-info" topic as TLS:Known(RPK/PSK)-MQTT:none.

- Expanded acronyms on their first use including the ones in the title.

- Added a definition for "Session".

- Corrected "CONNACK" definition, which earlier said it's the first packet sent by the broker.

- Added a statement that the the broker will disconnect on almost any error and may not keep session state.

- Clarified that the broker does not cache invalid tokens.

Version 04 to 05:

- Reorganised Section 2 such that "Unauthorised Request: Authorisation Server Discovery" is presented under Section 2.

- Fixed Figure 2 to remove the "empty" word.

- Clarified that MQTT v5.0 Brokers may implement username/password option for transporting the ACE token only for MQTT v.3.1.1 clients. This option is not recommended for MQTT v.5.0 clients.

- Changed Clean Session requirement both for MQTT v.5.0 and v.3.1.1. The Broker SHOULD NOT, instead of MUST NOT, continue sessions. Clarified expected behaviour if session continuation is supported. Added to the Security Considerations the potential misuse of session continuation.

- Fixed the Authentication Data to include token length for the Challenge/Response PoP.

- Added that Authorisation Server Discovery is triggered if a token is invalid and not only missing.

- Clarified that the Broker should not accept any other packets from Client after CONNECT and before sending CONNACK.

- Added that client reauthentication is accepted only for the challenge/response PoP.

- Added Ed25519 as mandatory to implement.

- Fixed typos.

Version 03 to 04:

- Linked the terms Broker and MQTT server more at the introduction of the document.

- Clarified support for MQTTv3.1.1 and removed phrases that might be considered as MQTTv5 is backwards compatible with MQTTv3.1.1

- Corrected the Informative and Normative references.

- For AS discovery, clarified the CONNECT message omits the Authentication Data field. Specified the User Property MUST be set to "ace_as_hint" for AS Request Creation Hints.

- Added that MQTT v5 brokers MAY also implement reduced interactions described for MQTTv3.1.1.

- Added to Section 3.1, in case of an authorisation failure and QoS level 0, the RS sends a DISCONNECT with reason code '0x87 (Not authorized)'.

- Added a pointer to section 4.7 of MQTTv5 spec for more information on topic names and filters.

- Added HS256 and RSA256 are mandatory to implement depending on the choice of symmetric or asymmetric validation.

- Added MQTT to the TLS exporter label to make it application specific: 'EXPORTER-ACE-MQTT-Sign-Challenge'.

- Added a format for Authentication Data so that length values prefix the token (or client nonce) when Authentication Data contains more than one piece of information.

- Clarified clients still connect over TLS (server-side) for the authz-info flow.

Version 02 to 03:

- Added the option of Broker certificate thumbprint in the 'rs_cnf' sent to the Client.

o Clarified the use of a random nonce from the TLS Exporter for PoP, added to the IANA requirements that the label should be registered.

o Added a client nonce, when Challenge/Response Authentication is used between Client and Broker.

o Clarified the use of the "authz-info" topic and the error response if token validation fails.

o Added clarification on wildcard use in scopes for publish/ subscribe permissions

o Reorganised sections so that token authorisation for publish/ subscribe messages are better placed.

Version 01 to 02:

o Clarified protection of Application Message payload as out of scope, and cited draft-palombini-ace-coap-pubsub-profile for a potential solution

o Expanded Client connection authorization to capture different options for Client and Broker authentication over TLS and MQTT

o Removed Payload (and specifically Client Identifier) from proof-of-possession in favor of using tls-exporter for a TLS-session based challenge.

o Moved token transport via "authz-info" topic from the Appendix to the main text.

o Clarified Will scope.

o Added MQTT AUTH to terminology.

o Typo fixes, and simplification of figures.

Version 00 to 01:

o Present the MQTTv5 as the RECOMMENDED version, and MQTT v3.1.1 for backward compatibility.

o Clarified Will message.

o Improved consistency in the use of terminology and upper/lower case.

- Defined Broker and MQTTS.

- Clarified HTTPS use for C-AS and RS-AS communication. Removed reference to actors document, and clarified the use of client authorization server.

- Clarified the Connect message payload and Client Identifier.

- Presented different methods for passing the token and PoP.

- Added new figures to explain AUTH packets exchange, updated CONNECT message figure.

Acknowledgements

The authors would like to thank Ludwig Seitz for his review and his input on the authorization information endpoint. The authors would like to thank Paul Fremantle for the initial discussions on MQTT v5.0 support.

Authors' Addresses

Cigdem Sengul
Brunel University
Dept. of Computer Science
Uxbridge UB8 3PH
UK

Email: csengul@acm.org

Anthony Kirby
Oxbotica
1a Milford House, Mayfield Road, Summertown
Oxford OX2 7EL
UK

Email: anthony@anthony.org

ACE
Internet-Draft
Intended status: Standards Track
Expires: April 8, 2021

M. Sahni, Ed.
S. Tripathi, Ed.
Palo Alto Networks
October 5, 2020

CoAP Transport for CMPV2
draft-msahni-ace-cmpv2-coap-transport-01

Abstract

This document specifies the use of Constrained Application Protocol (CoAP) as a transport medium for the Certificate Management Protocol Version 2 (CMPv2) and Lightweight CMP Profile [Lightweight-CMP-Profile] CMPv2 defines the interaction between various PKI entities for the purpose of certificate creation and management. CoAP is an HTTP like client-server protocol used by various constrained devices in IoT space.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Terminology 3
2. CoAP Transport For CMPv2 3
2.1. Discovery of CMP Entities 3
2.2. CoAP URI Format 3
2.3. CoAP Request Format 4
2.4. CoAP Content-Format 4
2.5. Announcement PKIMessage 4
2.6. CoAP Block Wise Transfer Mode 4
2.7. Multicast CoAP 4
3. Using CoAP over DTLS 4
4. Proxy support 5
4.1. CoAP to HTTP Proxy 5
4.2. CoAPs to HTTPs Proxy 5
5. Security Considerations 5
6. IANA Considerations 6
7. Acknowledgments 6
8. References 6
8.1. Normative References 6
8.2. Informative References 7
8.3. URL References 7
Authors' Addresses 7

1. Introduction

The CMPv2 is used by PKI entities for the generation and management of the certificates. One of the requirements of CMPv2 [RFC4210] is to be independent of the transport protocol in use. CMP has mechanisms to take care of required transactions, error reporting and encryption of messages. The CoAP defined in [RFC7252], [RFC7959] and [RFC8323] is a client-server protocol, like HTTP, that is designed to be used by constrained devices over constrained networks. The recommended transport for CoAP is UDP, however [RFC8323] specifies the support of CoAP over TCP, TLS and Websockets. This document specifies the use of CoAP as a transport medium for the CMPv2 and Lightweight CMP Profile [Lightweight-CMP-Profile]. This document, in general, follows the HTTP transport specifications for CMPv2 defined in [RFC6712] and specifies the additional requirements for CoAP transport. This document also provides guidance on how to use a "CoAP to HTTP" proxy for a better adaptation of CoAP transport without significant changes to the existing PKI entities. Although CoAP transport can be used for communication between Registration Authority (RA) and Certification Authority (CA) or between CAs, the scope of this document is for communication between End Entity (EE) and RA or EE and CA. This document is applicable only when the CoAP transport is being used for the CMPv2 transactions.

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.

2. CoAP Transport For CMPv2

CMPv2 transaction consists of passing PKIMesssage [RFC4210] between the PKI End Entities (EEs), Registration Authorities (RAs), and Certification Authorities (CAs). If the EEs are constrained devices then they will prefer, as a client, the use of CoAP instead of HTTP as a transport medium, while the RAs and CAs, in general, are not constrained and can support both CoAP and HTTP Client and Server implementation. This section specifes how to use CoAP as transport mechanism for CMPv2 or Lightweight CMP Profile [Lightweight-CMP-Profile].

2.1. Discovery of CMP Entities

The information about the URIs of CA and RA that is required by EEs can be either configured out of band on EEs or the EEs can use the service discovery mechanism described in section 7 of [RFC7252] to find them. The EE, RA SHOULD support service discovery as described in section 7 of [RFC7252]. An EE MUST verify the configured Root CA certificate against the Root CA certificate of the discovered entity to make sure it is talking to correct endpoint.

2.2. CoAP URI Format

The CoAP URI MUST follow the guidelines defined in section 3.6 of [RFC6712] for CMPv2 protocol. Implementations supporting the Lightweight CMP Profile [Lightweight-CMP-Profile] MUST follow the guidelines specified for HTTP transport defined in section 7.1 of Lightweight CMP Profile [Lightweight-CMP-Profile]. The URI’s for CoAP resources should start with coap:// instead of http:// and coaps:// instead of https://

2.3. CoAP Request Format

The CMPv2 PKIMessage MUST be DER encoded and sent as the body of the CoAP POST request. If the CoAP request is successful then the server should return a "2.05 Content" response code. If the CoAP request is not successful then an appropriate CoAP Client Error 4.xx or a Server Error 5.xx response code MUST be returned.

2.4. CoAP Content-Format

When transferring CMPv2 PKIMesssage over CoAP the media type application/pkixcmp MUST be used.

2.5. Announcement PKIMessage

When using the CoAP protocol, a PKI EE SHOULD poll for the possible changes via PKI Information request using General Message defined in the PKIMessage for various type of changes like CA key update or to get current CRL to check revocation or using Support messages defined in section 5.4 of Lightweight CMP Profile [Lightweight-CMP-Profile]. This will help constrained devices acting as EEs save resources as there is no need to open a listening socket for notifications and it will also make the use of a CoAP to HTTP proxy transparent to the EE.

2.6. CoAP Block Wise Transfer Mode

Since the CMPv2 PKIMesssage consists of a header body and optional fields a CMPv2 message can be much larger than the MTU of the outgoing interface of the device. In order to avoid IP fragmentation of messages that are exchanged between EEs and RAs or CAs, the Block Wise transfer [RFC7959] mode MUST be used for the CMPv2 Transactions over CoAP. If a CoAP to HTTP proxy is in the path between EEs and CA or EEs and RA then, it MUST receive the entire body from the client before sending the HTTP request to the server. This will avoid unnecessary errors in case the entire content of the PKIMesssage is not received and Proxy opens a connection with the server.

2.7. Multicast CoAP

CMPv2 PKIMessage request messages sent from EEs to RAs or from EEs to CAs over CoAP transport MUST not use a Multicast destination address.

3. Using CoAP over DTLS

Although CPMv2 protocol does not depend upon the underlying transport for the encryption and authentication of the messages but in cases when end to end secrecy is desired for the CoAP transport, CoAP over DTLS [RFC6347] as a transport medium SHOULD be used. Section 9.1 of

[RFC7252] defines how to use DTLS [RFC6347] for securing the CoAP. For CMPv2 and Lightweight CMP Profile [Lightweight-CMP-Profile] the clients should follow specifications defined in section 7.1 and section 7.2 of Lightweight CMP Profile [Lightweight-CMP-Profile] for setting up DTLS [RFC6347] connection either using certificates or shared secret. Once a DTLS [RFC6347] connection is established it SHOULD be used for as long as possible to avoid the frequent overhead of using DTLS [RFC6347] connection for constrained devices.

4. Proxy support

The use of a CoAP to HTTP proxy is recommended to avoid significant changes in the implementation of the CAs and RAs. However, if a proxy is in place then Announcements Messages cannot be passed to EEs efficiently. In case a CoAP to HTTP proxy is used for CMP transactions, it SHOULD support service discovery mentioned in section 2.1

4.1. CoAP to HTTP Proxy

If a CoAP to HTTP proxy is used then it MUST be positioned between EEs and RAs or between EEs and CAs when RA is not part of CMP transactions. The use of a CoAP to HTTP proxy between CAs and RAs is not recommended. The implementation of a CoAP to HTTP proxy is specified in Section 10 of [RFC7252]. The CoAP to HTTP proxy will also protect the CAs and RAs from UDP based Denial of Service attacks.

4.2. CoAPs to HTTPs Proxy

A CoAPs to HTTPS proxy (DTLS [RFC6347] transport to TLS [RFC8446] transport proxy) can be used instead of the CoAP to HTTP proxy if the server support HTTPS protocol, however client SHOULD be configured to trust the CA certificate used by proxy to sign the Man in the Middle (MITM) certificate for certificate chain validation [RFC5280].

5. Security Considerations

The CMPv2 protocol itself does not require secure transport and depends upon various mechanisms in the protocol itself to make sure that the transactions are secure. However, the CoAP protocol which uses UDP as layer 4 transport is vulnerable to many issues due to the connectionless characteristics of UDP itself. The Security considerations for CoAP protocol are mentioned in the [RFC7252]. Using a CoAP to HTTP proxy mitigates some of the risks as the requests from the EE’s can terminate inside the trusted network and will not require the server to listen on a UDP port making it safe from UDP based address spoofing, Denial of Service, and amplification attacks due to the characteristics of UDP.

6. IANA Considerations

This document requires a new entry to the CoAP Content-Formats Registry code for the content-type application/pkixcmp

7. Acknowledgments

The author would like to thank Hendrik Brockhaus, David von Oheimb, and Andreas Kretschmer for their guidance in writing the content of this document and providing valuable feedback.

8. References

8.1. 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>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6712] Kause, T. and M. Peylo, "Internet X.509 Public Key Infrastructure -- HTTP Transfer for the Certificate Management Protocol (CMP)", RFC 6712, DOI 10.17487/RFC6712, September 2012, <https://www.rfc-editor.org/info/rfc6712>.

[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>.

[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>.

[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>.

[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>.

8.2. Informative References

[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>.

[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>.

8.3. URL References

[Lightweight-CMP-Profile]
Brockhaus, H., Fries, S., and D. von Oheimb, "Lightweight CMP Profile", 2020, <https://tools.ietf.org/html/draft-brockhaus-lamps-lightweight-cmp-profile-03>.

Authors' Addresses

Mohit Sahni (editor)
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
US

EMail: msahni@paloaltonetworks.com

Saurabh Tripathi (editor)
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
US

EMail: stripathi@paloaltonetworks.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
R. Hoeglund
RISE AB
L. Seitz
Combitech
F. Palombini
Ericsson AB

November 02, 2020

Group OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework
draft-tiloca-ace-group-oscore-profile-04

Abstract

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework. The profile uses Group OSCORE to provide communication security between a Client and a (set of) Resource Server(s) as members of an OSCORE Group. The profile securely binds an OAuth 2.0 Access Token with the public key of the Client associated to the signing private key used in the OSCORE group. The profile uses Group OSCORE to achieve server authentication, as well as proof-of-possession for the Client's public key. Also, it provides proof of the Client's membership to the correct OSCORE group, by binding the Access Token to information from the Group OSCORE Security Context, thus allowing the Resource Server(s) to verify the Client's membership upon receiving a message protected with Group OSCORE from the Client.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction   3
  1.1. Terminology   6
2. Protocol Overview   6
  2.1. Pre-Conditions   9
  2.2. Access Token Retrieval   9
  2.3. Access Token Posting 10
  2.4. Secure Communication 10
3. Client-AS Communication 11
  3.1. C-to-AS: POST to Token Endpoint 11
    3.1.1. 'context_id' Parameter 13
    3.1.2. 'salt_input' Parameter 13
    3.1.3. 'client_cred_verify' Parameter 13
  3.2. AS-to-C: Access Token 14
    3.2.1. Salt Input Claim 17
    3.2.2. Context ID Input Claim 17
4. Client-RS Communication 17
  4.1. C-to-RS POST to authz-info Endpoint 18
  4.2. RS-to-C: 2.01 (Created) 18
  4.3. Client-RS Secure Communication 19
    4.3.1. Client Side 19
    4.3.2. Resource Server Side 19
  4.4. Access Rights Verification 20
5. Secure Communication with the AS 20
6. Discarding the Security Context 21
7. CBOR Mappings 21
8. Security Considerations 21
9. Privacy Considerations 22
10. IANA Considerations 23
  10.1. ACE Profile Registry 23
  10.2. OAuth Parameters Registry 23
  10.3. OAuth Parameters CBOR Mappings Registry 24

10.4. CBOR Web Token Claims Registry 25
10.5. TLS Exporter Label Registry 26
11. References 26
11.1. Normative References 26
11.2. Informative References 28
Appendix A. Dual Mode (Group OSCORE & OSCORE) 29
A.1. Protocol Overview 30
A.1.1. Pre-Conditions 32
A.1.2. Access Token Posting 32
A.1.3. Setup of the Pairwise OSCORE Security Context 33
A.1.4. Secure Communication 34
A.2. Client-AS Communication 34
A.2.1. C-to-AS: POST to Token Endpoint 35
A.2.2. AS-to-C: Access Token 37
A.3. Client-RS Communication 44
A.3.1. C-to-RS POST to authz-info Endpoint 45
A.3.2. RS-to-C: 2.01 (Created) 46
A.3.3. OSCORE Setup - Client Side 46
A.3.4. OSCORE Setup - Resource Server Side 49
A.3.5. Access Rights Verification 51
A.4. Secure Communication with the AS 51
A.5. Discarding the Security Context 51
A.6. CBOR Mappings 52
A.7. Security Considerations 52
A.8. Privacy Considerations 52
Appendix B. Profile Requirements 53
Acknowledgments 54
Authors' Addresses 54

1. Introduction

A number of applications rely on a group communication model, where a Client can access a resource shared by multiple Resource Servers at once, e.g. over IP multicast. Typical examples are switching of luminaries, actuators control, and distribution of software updates. Secure communication in the group can be achieved by sharing a set of key material, which is typically provided upon joining the group.

For some of such applications, it may be just fine to enforce access control in a straightforward fashion. That is, any Client authorized to join the group, hence to get the group key material, can be also implicitly authorized to perform any action at any resource of any Server in the group. An example of application where such implicit authorization might be used is a simple lighting scenario, where the lightbulbs are the Servers, while the user account on an app on the user's phone is the Client. In this case, it might be fine to not require additional authorization evidence from any user account, if it is acceptable that any current group member is also authorized to switch on and off any light, or to check their status.

However, in different instances of such applications, the approach above is not desirable, as different group members are intended to have different access rights to resources of other group members. That is, access control to the secure group communication channel and access control to the resource space provided by servers in the group should remain logically separated domains. For instance, a more fine-grained approach is required in the two following use cases.

As a first case, an application provides control of smart locks acting as Servers in the group, where: a first type of Client, e.g. a user account of a child, is allowed to only query the status of the smart locks; while a second type of Client, e.g. a user account of a parent, is allowed to both query and change the status of the smart locks. Further similar applications concern the enforcement of different sets of permissions in groups with sensor/actuator devices, e.g. thermostats, acting as Servers. Also, some group members may even be intended as Servers only. Hence, they must be prevented from acting as Clients altogether and from accessing resources at other Servers, especially when attempting to perform non-safe operations.

As a second case, building automation scenarios often rely on Servers that, under different circumstances, enforce different level of priority for processing received commands. For instance, BACnet deployments consider multiple classes of Clients, e.g. a normal light switch (C1) and an emergency fire panel (C2). Then, a C1 Client is not allowed to override a command from a C2 Client, until the latter relinquishes control at its higher priority. That is: i) only C2 Clients should be able to adjust the minimum required level of priority on the Servers, so rightly locking out C1 Clients if needed; and ii) when a Server is set to accept only high-priority commands, only C2 Clients should be able to perform such commands otherwise allowed also to C1 Clients. Given the different maximum authority of different Clients, fine-grained access control would effectively limit the execution of high- and emergency-priority commands only to devices that are in fact authorized to do so. Besides, it would prevent a misconfigured or compromised device from initiating a high-priority command and lock out normal control.

In the cases above, being a legitimate group member and owning the group key material is not supposed to imply any particular access rights. Also, introducing a different security group for each different set of access rights would result in additional key material to distribute and manage. In particular, if the access rights for a single node change, this would require to evict that node from the current group, followed by that node joining a different group aligned with its new access rights. Moreover, the key material of both groups would have to be renewed for their current members. Overall, this would have a non negligible impact on operations and performance in the system.

A fine-grained access control model can be rather enforced within a same group, by using the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. That is, a Client has to first obtain authorization credentials in the form of an Access Token, and post it to the Resource Server(s) in the group before accessing the intended resources.

The ACE framework delegates to separate profile documents how to secure communications between the Client and the Resource Server. However each of the current profiles of ACE defined in [I-D.ietf-ace-oscore-profile] [I-D.ietf-ace-dtls-authorize] [I-D.ietf-ace-mqtt-tls-profile] admits a single security protocol that cannot be used to protect group messages sent over IP multicast.

This document specifies a profile of ACE, where a Client uses CoAP [RFC7252] or CoAP over IP multicast [I-D.ietf-core-groupcomm-bis] to communicate to one or multiple Resource Servers, which are members of an application group and share a common set of resources. This profile uses Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the security protocol to protect messages exchanged between the Client and the Resource Servers. Hence, it requires that both the Client and the Resource Servers have previously joined the same OSCORE group.

That is, this profile describes how access control is enforced for a Client after it has joined an OSCORE group, to access resources at other members in that group. The process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] takes place before the process described in this document, and is out of the scope of this profile.

The Client proves its access to be authorized to the Resource Server by using an Access Token, which is bound to a key (the proof-of-possession key). This profile uses Group OSCORE to achieve server authentication, as well as proof-of-possession for the Client's public key associated to the signing private key used in an OSCORE group. Note that the proof of possession is not done by a dedicated protocol element, but rather occurs after the first Group OSCORE exchange. Furthermore, this profile provides proof of the Client's membership to the correct OSCORE group, by binding the Access Token to the Client's public key and information from the pre-established Group OSCORE Security Context, thus allowing the Resource Server to verify this upon reception of a messages protected with Group OSCORE from the Client.

OSCORE [RFC8613] specifies how to use COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] to secure CoAP messages. Group OSCORE builds on OSCORE to provide secure group communication, and ensures source authentication either: by means of digital counter signatures embedded in protected messages (in group mode); by protecting messages with pairwise key material derived from the asymmetric keys of the two peers exchanging the message (in pairwise mode).

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 related to CBOR [I-D.ietf-cbor-7049bis], COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], CoAP [RFC7252], OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications among members are secured with Group OSCORE.

Readers are expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz], as well as in the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

2. Protocol Overview

This section provides an overview of this profile, i.e. on how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to secure communications between a Client and a (set of) Resource Server(s) using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Note that this profile of ACE describes how access control can be enforced for a node after it has joined an OSCORE group, to access resources at other members in that group.

In particular, the process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] must take place before the process described in this document, and is out of the scope of this profile.

An overview of the protocol flow for this profile is shown in Figure 1. In the figure, it is assumed that both RS1 and RS2 are associated with the same AS. It is also assumed that C, RS1 and RS2 have previously joined an OSCORE group with Group Identifier (gid) "abcd0000", and got assigned Sender ID (sid) "0", "1" and "2" in the group, respectively. The names of messages coincide with those of [I-D.ietf-ace-oauth-authz] when applicable.

```
C                            RS1          RS2                         AS
| [--- Resource Request --->] |            |                           |
|                             |            |                           |
| [<----  AS Request ------]  |            |                           |
|       Creation Hints        |            |                           |
|                             |            |                           |
|-------- POST /token ------------------------------------------------>|
|  (aud: RS1, sid: 0, gid: abcd0000, ...)  |                           |
|                             |            |                           |
|<--------------------------------- Access Token + RS Information -----|
|                             | (aud: RS1, sid: 0, gid: abcd0000, ...) |
|                             |            |                           |
|---- POST /authz-info ------>|            |                           |
|      (access_token)         |            |                           |
|                             |            |                           |
|<--- 2.01 Created -----------|            |                           |
|                             |            |                           |
|-------- POST /token ------------------------------------------------>|
|  (aud: RS2, sid: 0, gid: abcd0000, ...)  |                           |
|                             |            |                           |
|<--------------------------------- Access Token + RS Information -----|
|                             | (aud: RS2, sid: 0, gid: abcd0000, ...) |
|                             |            |                           |
|----- POST /authz-info ------------------>|                           |
|       (access_token)        |            |                           |
|                             |            |                           |
|<--- 2.01 Created ------------------------|                           |
|                             |            |                           |
|-- Group OSCORE Request -+-->|            |                           |
| (kid: 0, gid: abcd0000)  \-------------->|                           |
|                             |            |                           |
|                          /proof-of-possession/                       |
|                             |            |                           |
|<--- Group OSCORE Response --|            |                           |
|          (kid: 1)           |            |                           |
|                             |            |                           |
/proof-of-possession/         |            |                           |
|                             |            |                           |
|<--- Group OSCORE Response ---------------|                           |
|          (kid: 2)           |            |                           |
|                             |            |                           |
/proof-of-possession/         |            |                           |
|             ...             |            |                           |
```

Figure 1: Protocol Overview.

2.1. Pre-Conditions

Using Group OSCORE and this profile requires both the Client and the Resource Servers to have previously joined the same OSCORE group. This especially includes the derivation of the Group OSCORE Security Context and the assignment of unique Sender IDs to use in the group. Nodes may join the OSCORE group through the respective Group Manager by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore], which is also based on ACE.

After the Client and Resource Servers have joined the group, this profile provides access control for accessing resources on those Resource Servers, by securely communicating with Group OSCORE.

As a pre-requisite for this profile, the Client has to have successfully joined the OSCORE group where also the Resource Servers (RSs) are members. Depending on the limited information initially available, the Client may have to first discover the exact OSCORE group used by the RSs for the resources of interest, e.g. by using the approach defined in [I-D.tiloca-core-oscore-discovery].

2.2. Access Token Retrieval

This profile requires that the Client retrieves an Access Token from the AS for the resource(s) it wants to access on each of the RSs, using the /token endpoint, as specified in Section 5.6 of [I-D.ietf-ace-oauth-authz]. In a general case, it can be assumed that different RSs are associated to different ASs, even if the RSs are members of a same OSCORE group.

In the Access Token request to the AS, the Client MUST include the Group Identifier of the OSCORE group and its own Sender ID in that group. The AS MUST specify these pieces of information in the Access Token, included in the Access Token response to the Client.

Furthermore, in the Access Token request to the AS, the Client MUST also include: its own public key, associated to the private signing key used in the OSCORE group; and a signature computed with such private key, over a quantity uniquely related to the secure communication association between the Client and the AS. The AS MUST include also the public key indicated by the Client in the Access Token.

The Access Token request and response MUST be confidentiality-protected and ensure authenticity. This profile RECOMMENDS the use of OSCORE between the Client and the AS, but TLS [RFC5246][RFC8446] or DTLS [RFC6347][I-D.ietf-tls-dtls13] MAY be used additionally or instead.

2.3. Access Token Posting

After having retrieved the Access Token from the AS, the Client posts the Access Token to the RS, using the /authz-info endpoint and mechanisms specified in Section 5.8 of [I-D.ietf-ace-oauth-authz], as well as Content-Format = application/ace+cbor. When using this profile, the communication with the /authz-info endpoint is not protected.

If the Access Token is valid, the RS replies to this POST request with a 2.01 (Created) response with Content-Format = application/ace+cbor. Also, the RS associates the received Access Token with the Group OSCORE Security Context identified by the Group Identifier specified in the Access Token, following Section 3.2 of [RFC8613]. In practice, the RS maintains a collection of Security Contexts with associated authorization information, for all the clients that it is currently communicating with, and the authorization information is a policy used as input when processing requests from those clients.

Finally, the RS stores the association between i) the authorization information from the Access Token; and ii) the Group Identifier of the OSCORE group together with the Sender ID and the public key of the Client in that group. This binds the Access Token with the Group OSCORE Security Context of the OSCORE group.

Finally, when the Client communicates with the RS using the Group OSCORE Security Context, the RS verifies that the Client is a legitimate member of the OSCORE group and especially the exact group member with the same Sender ID associated to the Access Token. This occurs when verifying a request protected with Group OSCORE, since it embeds a counter signature computed also over the Client's Sender ID included in the message.

2.4. Secure Communication

The Client can send a request protected with Group OSCORE [I-D.ietf-core-oscore-groupcomm] to the RS. This can be a unicast request addressed to the RS, or a multicast request addressed to the OSCORE group where the RS is also a member. To this end, the Client uses the Group OSCORE Security Context already established upon joining the OSCORE group, e.g. by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. The RS may send a response back to the Client, protecting it by means of the same Group OSCORE Security Context.

3. Client-AS Communication

This section details the Access Token POST Request that the Client sends to the /token endpoint of the AS, as well as the related Access Token response.

The Access Token MUST be bound to the public key of the client as proof-of-possession key (pop-key), by means of the 'cnf' claim.

3.1. C-to-AS: POST to Token Endpoint

The Client-to-AS request is specified in Section 5.6.1 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.

The POST request is formatted as the analogous Client-to-AS request in the OSCORE profile of ACE (see Section 3.1 of [I-D.ietf-ace-oscore-profile]), with the following additional parameters that MUST be included in the payload.

o 'context_id', defined in Section 3.1.1 of this specification. This parameter specifies the Group Identifier (GID), i.e. the Id Context of an OSCORE group where the Client and the RS are currently members. In particular, the Client wishes to communicate with the RS using the Group OSCORE Security Context associated to that OSCORE group.

o 'salt_input', defined in Section 3.1.2 of this specification. This parameter includes the Sender ID that the Client has in the OSCORE group whose GID is specified in the 'context_id' parameter above.

o 'req_cnf', defined in Section 3.1 of [I-D.ietf-ace-oauth-params]. This parameter includes the public key associated to the signing private key that the Client uses in the OSCORE group whose GID is specified in the 'context_id' parameter above. This public key will be used as the pop-key bound to the Access Token.

o 'client_cred_verify', defined in Section 3.1.3 of this specification. This parameter includes a signature computed by the Client, by using the private key associated to the public key in the 'req_cnf' parameter above. This allows the AS to verify that the Client indeed owns the private key associated to that public key, as its alleged identity credential within the OSCORE group. The information to be signed MUST be the byte representation of a quantity that uniquely represents the secure communication association between the Client and the AS. It is

RECOMMENDED that the Client considers the following as information to sign.

* If the Client and the AS communicate over (D)TLS, the information to sign is an exporter value computed as defined in Section 7.5 of [RFC8446]. In particular, the exporter label MUST be 'EXPORTER-ACE-Sign-Challenge-Client-AS' defined in Section 10.5 of this specification, together with an empty 'context_value', and 32 bytes as 'key_length'.

* If the Client and the AS communicate over OSCORE, the information to sign is the output PRK of a HKDF-Extract step [RFC5869], i.e. PRK = HMAC-Hash(salt, IKM). In particular, 'salt' takes (x1 | x2), where x1 is the ID Context of the OSCORE Security Context between the Client and the AS, x2 is the Sender ID of the Client in that Context, and | denotes byte string concatenation. Also, 'IKM' is the OSCORE Master Secret of the OSCORE Security Context between the Client and the AS. The HKDF MUST be one of the HMAC-based HKDF [RFC5869] algorithms defined for COSE [I-D.ietf-cose-rfc8152bis-algs]. HKDF SHA-256 is mandatory to implement.

An example of such a request, with payload in CBOR diagnostic notation without the tag and value abbreviations is reported 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",
          "context_id" : h'abcd0000',
          "salt_input" : h'00',
          "req_cnf" : {
            "COSE_Key" : {
              "kty" : EC2,
              "crv" : P-256,
              "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
                      27c9e354089bbe13',
              "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
                      731e79a3b4e47120'
            }
          },
          "client_cred_verify" : h'...'
          (signature content omitted for brevity)
        }
```

Figure 2: Example C-to-AS POST /token request for an Access Token bound to an asymmetric key.

3.1.1. 'context_id' Parameter

The 'context_id' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.6.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides a value that the Client wishes to use with the RS as a hint for a security context. Its exact content is profile specific.

3.1.2. 'salt_input' Parameter

The 'salt_input' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.6.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides a value that the Client wishes to use as part of a salt with the RS, for deriving cryptographic key material. Its exact content is profile specific.

3.1.3. 'client_cred_verify' Parameter

The 'client_cred_verify' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.6.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides a signature computed by the Client to prove the possession of its own private key.

3.2. AS-to-C: Access Token

After having verified 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 MUST verify the signature in the 'client_cred_verify' parameter, by using the public key specified in the 'req_cnf' parameter. If the verification fails, the AS considers the Client request invalid.

If all verifications are successful, the AS responds as defined in Section 5.6.2 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 5.6.3 of [I-D.ietf-ace-oauth-authz].

The AS can signal that the use of Group OSCORE is REQUIRED for a specific Access Token by including the 'profile' parameter with the value "coap_group_oscore" in the Access Token response. The Client MUST use Group OSCORE towards all the Resource Servers for which this Access Token is valid. Usually, it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile negotiation can be omitted.

The AS MUST include the following information as metadata of the issued Access Token. The use of CBOR web tokens (CWT) as specified in [RFC8392] is RECOMMENDED.

o The same parameter 'profile' included in the Token Response to the Client.

o The salt input specified in the 'salt_input' parameter of the Token Request. If the Access Token is a CWT, the content of the 'salt_input' parameter MUST be placed in the 'salt_input' claim of the Access Token, defined in Section 3.2.1 of this specification.

o The Context Id input specified in the 'context_id' parameter of the Token Request. If the Access Token is a CWT, the content of the 'context_id' parameter MUST be placed in the 'contextId_input' claim of the Access Token, defined in Section 3.2.2 of this specification.

o The public key that the client uses in the OSCORE group and specified in the 'req_cnf' parameter of the Token request. If the Access Token is a CWT, the public key MUST be specified in the 'cnf' claim, which follows the syntax from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. Alternative Value Types defined in future specifications are fine to consider if indicating a non-encrypted asymmetric key.

Figure 3 shows an example of such an AS response, with payload in CBOR diagnostic notation without the tag and value abbreviations.

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

Figure 3: Example AS-to-C Access Token response with the Group OSCORE profile.

Figure 4 shows an example CWT Claims Set, containing the Client's public key in the group (as pop-key) in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```
     {
       "aud" : "tempSensorInLivingRoom",
       "iat" : "1360189224",
       "exp" : "1360289224",
       "scope" :  "temperature_g firmware_p",
       "cnf" : {
         "COSE_Key" : {
           "kty" : EC2,
           "crv" : P-256,
           "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
                   27c9e354089bbe13',
           "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
                   731e79a3b4e47120'
       },
       "salt_input" : h'00',
       "contextId_input" : h'abcd0000'
     }
```

Figure 4: Example CWT Claims Set with OSCORE parameters (CBOR diagnostic notation).

The same CWT Claims Set as in Figure 4 and encoded in CBOR is shown in Figure 5, using the value abbreviations defined in [I-D.ietf-ace-oauth-authz] and [RFC8747]. 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: 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.

```
A7                                     # map(7)
   03                                  # unsigned(3)
   76                                  # text(22)
      74656D7053656E736F72496E4C6976696E67526F6F6D
   06                                  # unsigned(6)
   1A 5112D728                         # unsigned(1360189224)
   04                                  # unsigned(4)
   1A 51145DC8                         # unsigned(1360289224)
   09                                  # unsigned(9)
   78 18                               # text(24)
      74656D706572617475726655F67206669726D776172655F70
   08                                  # unsigned(8)
   A1                                  # map(1)
      01                               # unsigned(1)
      A4                               # map(4)
         01                            # unsigned(1)
         02                            # unsigned(2)
         20                            # negative(0)
         01                            # unsigned(1)
         21                            # negative(1)
         58 20                         # bytes(32)
            D7CC072DE2205BDC1537A543D53C60A6ACB62ECCD890C7FA27C9
            E354089BBE13
         22                            # negative(2)
         58 20                         # bytes(32)
            F95E1D4B851A2CC80FFF87D8E23F22AFB725D535E515D020731E
            79A3B4E47120
   18 3C                               # unsigned(60)
   41                                  # bytes(1)
      00
   18 3D                               # unsigned(61)
   44                                  # bytes(4)
      ABCD0000
```

Figure 5: Example CWT Claims Set with OSCORE parameters, CBOR encoded.

3.2.1. Salt Input Claim

The 'salt_input' claim provides a value that the Client requesting the Access Token wishes to use as a part of a salt with the RS, e.g. for deriving cryptographic material.

This parameter specifies the value of the salt input, encoded as a CBOR byte string.

3.2.2. Context ID Input Claim

The 'contextId_input' claim provides a value that the Client requesting the Access Token wishes to use with the RS, as a hint for a security context.

This parameter specifies the value of the Context ID input, encoded as a CBOR byte string.

4. Client-RS Communication

This section details the POST request and response to the /authz-info endpoint between the Client and the RS.

The proof-of-possession required to bind the Access Token to the Client is explicitly performed when the RS receives and verifies a request from the Client protected with Group OSCORE, either with the group mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]) or with the pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

In particular, the RS uses the Client's public key bound to the Access Token, either when verifying the counter signature of the request (if protected with the group mode), or when verifying the request as integrity-protected with pairwise key material derived from the two peers' asymmetric keys (if protected with the pairwise mode). In either case, the RS also authenticates the Client.

Similarly, when receiving a protected response from the RS, the Client uses the RS's public key either when verifying the counter signature of the response (if protected with the group mode), or when verifying the response as integrity-protected with pairwise key material derived from the two peers' asymmetric keys (if protected with the pairwise mode). In either case, the Client also authenticates the RS. Mutual authentication is only achieved after the client has successfully verified the Group OSCORE protected response from the RS.

Therefore, an attacker using a stolen Access Token cannot generate a valid Group OSCORE message signed with the Client's private key, and thus cannot prove possession of the pop-key bound to the Access Token. Also, if a Client legitimately owns an Access Token but has not joined the OSCORE group, it cannot generate a valid Group OSCORE message, as it does not own the necessary key material shared among the group members.

Furthermore, a Client C1 is supposed to obtain a valid Access Token from the AS, as including the public key associated to its own signing key used in the OSCORE group, together with its own Sender ID in that OSCORE group (see Section 3.1). This makes it possible for the RS receiving an Access Token to verify with the Group Manager of that OSCORE group whether such a Client has indeed that Sender ID and that public key in the OSCORE group.

As a consequence, a different Client C2, also member of the same OSCORE group, is not able to impersonate C1, by: i) getting a valid Access Token, specifying the Sender ID of C1 and a different (made-up) public key; ii) successfully posting the Access Token to RS; and then iii) attempting to communicate using Group OSCORE impersonating C1, while blaming C1 for the consequences.

4.1. C-to-RS POST to authz-info Endpoint

The Client posts the Access Token to the /authz-info endpoint of the RS, as defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz].

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

The RS MUST verify the validity of the Access Token as defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz], with the following additions.

- o The RS checks that the claims 'salt_input', 'contextId_input' and 'cnf' are included in the Access Token.

- o The RS considers the content of the 'cnf' claim as the public key associated to the signing private key of the Client in the OSCORE group, whose GID is specified in the 'contextId_input' claim above. If it does not already store that public key, the RS MUST request it to the Group Manager of the OSCORE group as described in [I-D.ietf-ace-key-groupcomm-oscore], specifying the Sender ID of that Client in the OSCORE group, i.e. the value of the 'salt_input' claim above. The RS MUST check that the key retrieved from the Group Manager matches the one retrieved from the 'cnf' claim. When doing so, the 'kid' parameter of the COSE_Key, if present, MUST NOT be considered for the comparison.

If any of the checks above fails, the RS MUST consider the Access Token non valid, and MUST respond to the Client with an error response code equivalent to the CoAP code 4.00 (Bad Request).

If the Access Token is valid and further checks on its content are successful, the RS associates the authorization information from the Access Token with the Group OSCORE Security Context.

In particular, the RS associates the authorization information from the Access Token with the tuple (GID, SaltInput, PubKey), where GID is the Group Identifier of the OSCORE Group, while SaltInput and PubKey are the Sender ID and the public key that the Client uses in that OSCORE group, respectively. These can be retrieved from the 'contextId_input', 'salt_input' and 'cnf' claims of the Access Token, respectively. The RS MUST keep this association up-to-date over time.

Finally, the RS MUST send a 2.01 (Created) response to the Client, as defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz].

4.3. Client-RS Secure Communication

When previously joining the OSCORE group, both the Client and RS have already established the related Group OSCORE Security Context to communicate as group members. Therefore, they can simply start to securely communicate using Group OSCORE, without deriving any additional key material or security association.

4.3.1. Client Side

After having received the 2.01 (Created) response from the RS, following the POST request to the authz-info endpoint, the Client can start to communicate with the RS using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

When communicating with the RS to access the resources as specified by the authorization information, the Client MUST use the Group OSCORE Security Context of the OSCORE group, whose GID was specified in the 'context_id' parameter of the Token request.

4.3.2. Resource Server Side

After successful validation of the Access Token as defined in Section 4.2 and after having sent the 2.01 (Created) response, the RS can start to communicate with the Client using Group OSCORE [I-D.ietf-core-oscore-groupcomm]. Additionally, for every incoming request, if Group OSCORE verification succeeds, the verification of access rights is performed as described in Section 4.4.

After the expiration of the Access Token related to a Group OSCORE Security Context, if the Client uses the Group OSCORE Security Context to send a request for any resource intended for OSCORE group members and that requires an active Access Token, the RS MUST respond with a 4.01 (Unauthorized) error message protected with the Group OSCORE Security 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 a Group OSCORE-protected request from a Client, the RS processes it according to [I-D.ietf-core-oscore-groupcomm].

If the Group OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the authorization information from the Access Token associated to the Group OSCORE Security Context. Then, the RS MUST verify that the action requested on the resource is authorized.

The response code MUST be 4.01 (Unauthorized) if the RS has no valid Access Token for the Client. If the RS has an Access Token for the Client but no actions are authorized on the target resource, the RS MUST reject the request with a 4.03 (Forbidden). If the RS has an Access Token for the Client but the requested action is not authorized, the RS MUST reject the request with a 4.05 (Method Not Allowed).

5. Secure Communication with the AS

As specified in the ACE framework (Section 5.7 of [I-D.ietf-ace-oauth-authz]), the requesting entity (RS and/or Client) and the AS communicate via the /introspection or /token endpoint. The use of CoAP and OSCORE for this communication is RECOMMENDED in this profile. Other protocols (such as HTTP and DTLS or TLS) MAY be used instead.

If OSCORE [RFC8613] is used, the requesting entity and the AS are expected to have a pre-established Security Context in place. How this Security Context is established is out of the scope of this profile. Furthermore, the requesting entity and the AS communicate using OSCORE through the /introspection endpoint as specified in Section 5.7 of [I-D.ietf-ace-oauth-authz], and through the /token endpoint as specified in Section 5.6 of [I-D.ietf-ace-oauth-authz].

6. Discarding the Security Context

As members of an OSCORE group, the Client and the RS may independently leave the group or be forced to, e.g. if compromised or suspected so. Upon leaving the OSCORE group, the Client or RS also discards the Group OSCORE Security Context, which may anyway be renewed by the Group Manager through a group rekeying process (see Section 3.1 of [I-D.ietf-core-oscore-groupcomm]).

The Client or RS can acquire a new Group OSCORE Security Context, by re-joining the OSCORE group, e.g. by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. In such a case, the Client SHOULD request a new Access Token and post it to the RS.

7. CBOR Mappings

The new parameters defined in this document MUST be mapped to CBOR types as specified in Figure 6, using the given integer abbreviation for the map key.

Parameter name	CBOR Key	Value Type
context_id	TBD1	bstr
salt_input	TBD2	bstr
client_cred_verify	TBD3	bstr

Figure 6: CBOR mappings for new parameters.

The new claims defined in this document MUST be mapped to CBOR types as specified in Figure 7, using the given integer abbreviation for the map key.

Claim name	CBOR Key	Value Type
salt_input	TBD4	bstr
contextId_input	TBD5	bstr

Figure 7: CBOR mappings for new claims.

8. 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 ACE framework also apply to this profile.

This specification inherits the general security considerations about Group OSCORE [I-D.ietf-core-oscore-groupcomm], as to the specific use of Group OSCORE according to this profile.

Group OSCORE is designed to secure point-to-point as well as point-to-multipoint communications, providing a secure binding between a single request and multiple corresponding responses. In particular, Group OSCORE fulfills the same security requirements of OSCORE, for group requests and responses.

Group OSCORE ensures source authentication of messages both in group mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]) and in pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

When protecting an outgoing message in group mode, the sender uses its private key to compute a digital counter signature, which is embedded in the protected message. The group mode can be used to protect messages sent over multicast to multiple recipients, or sent over unicast to one recipient.

When protecting an outgoing message in pairwise mode, the sender uses a pairwise symmetric key, as derived from the asymmetric keys of the two peers exchanging the message. The pairwise mode can be used to protect only messages sent over unicast to one recipient.

9. 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 ACE framework also apply to this profile.

As this profile uses Group OSCORE, the privacy considerations from [I-D.ietf-core-oscore-groupcomm] apply to this document as well.

An unprotected response to an unauthorized request may disclose information about the RS and/or its existing relationship with the Client. It is advisable to include as little information as possible in an unencrypted response. However, since both the Client and the RS share a Group OSCORE Security Context, unauthorized, yet protected requests are followed by protected responses, which can thus include more detailed information.

Although it may be encrypted, the Access Token is sent in the clear to the /authz-info endpoint at the RS. Thus, if the Client uses the same single Access Token from multiple locations with multiple Resource Servers, it can risk being tracked through the Access Token's value.

Note that, even though communications are protected with Group OSCORE, some information might still leak, due to the observable size, source address and destination address of exchanged messages.

10. IANA Considerations

This document has the following actions for IANA.

10.1. ACE Profile Registry

IANA is asked to enter the following value into the "ACE Profile" Registry defined in Section 8.8 of [I-D.ietf-ace-oauth-authz].

- o Name: coap_group_oscore
- o Description: Profile to secure communications between constrained nodes using the Authentication and Authorization for Constrained Environments framework, by enabling authentication and fine-grained authorization of members of an OSCORE group, that use a pre-established Group OSCORE Security Context to communicate with Group OSCORE. Optionally, the dual mode defined in Appendix A additionally establishes a pairwise OSCORE Security Context, and thus also enables OSCORE communication between two members of the OSCORE group.
- o CBOR Value: TBD (value between 1 and 255)
- o Reference: [[this document]]

10.2. OAuth Parameters Registry

IANA is asked to enter the following values into the "OAuth Parameters" Registry defined in Section 11.2 of [RFC6749].

- o Name: "context_id"
- o Parameter Usage Location: token request
- o Change Controller: IESG
- o Specification Document(s): Section 3.1.1 of [[this document]]

- o Name: "salt_input"

- Parameter Usage Location: token request
- Change Controller: IESG
- Specification Document(s): Section 3.1.2 of [[this document]]

- Name: "client_cred_verify"
- Parameter Usage Location: token request
- Change Controller: IESG
- Specification Document(s): Section 3.1.3 of [[this document]]

- Name: "client_cred"
- Parameter Usage Location: token request
- Change Controller: IESG
- Specification Document(s): Appendix A.2.1.1 of [[this document]]

10.3. OAuth Parameters CBOR Mappings Registry

IANA is asked to enter the following values into the "OAuth Parameters CBOR Mappings" Registry defined in Section 8.10 of [I-D.ietf-ace-oauth-authz].

- Name: "context_id"
- CBOR Key: TBD1
- Value Type: bstr
- Reference: Section 3.1.1 of [[this document]]

- Name: "salt_input"
- CBOR Key: TBD2
- Value Type: bstr
- Reference: Section 3.1.2 of [[this document]]

- Name: "client_cred_verify"
- CBOR Key: TBD3
- Value Type: bstr
- Reference: Section 3.1.3 of [[this document]]

- Name: "client_cred"
- CBOR Key: TBD6
- Value Type: bstr
- Reference: Appendix A.2.1.1 of [[this document]]

10.4. CBOR Web Token Claims Registry

IANA is asked to enter the following values into the "CBOR Web Token Claims" Registry defined in Section 9.1 of [RFC8392].

- Claim Name: "salt_input"
- Claim Description: Client provided salt input
- JWT Claim Name: "N/A"
- Claim Key: TBD4
- Claim Value Type(s): bstr
- Change Controller: IESG
- Specification Document(s): Section 3.2.1 of [[this document]]

- Claim Name: "contextId_input"
- Claim Description: Client context id input
- JWT Claim Name: "N/A"
- Claim Key: TBD5
- Claim Value Type(s): bstr
- Change Controller: IESG

- Specification Document(s): Section 3.2.2 of [[this document]]

- Claim Name: "client_cred"

- Claim Description: Client Credential

- JWT Claim Name: "N/A"

- Claim Key: TBD7

- Claim Value Type(s): map

- Change Controller: IESG

- Specification Document(s): Appendix A.2.2.2 of [[this document]]

10.5. TLS Exporter Label Registry

IANA is asked to register the following entry in the "TLS Exporter Label" Registry defined in Section 6 of [RFC5705] and updated in Section 12 of [RFC8447].

- Value: EXPORTER-ACE-Sign-Challenge-Client-AS

- DTLS-OK: Y

- Recommended: N

- Reference: [[this document]] (Section 3.1)

11. References

11.1. Normative References

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-09 (work in progress), November 2020.

[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-35 (work in progress), June 2020.

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-13 (work in progress), April 2020.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-13 (work in progress), October 2020.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-ietf-core-groupcomm-bis-02 (work in progress), November 2020.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-10 (work in progress), November 2020.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-12 (work in progress), September 2020.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-14 (work in progress), September 2020.

[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>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[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>.

[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>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[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>.

[RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

[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>.

[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>.

11.2. Informative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-14 (work in progress), October 2020.

[I-D.ietf-ace-mqtt-tls-profile]
Sengul, C. and A. Kirby, "Message Queuing Telemetry Transport (MQTT)-TLS profile of Authentication and Authorization for Constrained Environments (ACE) Framework", draft-ietf-ace-mqtt-tls-profile-08 (work in progress), November 2020.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-07 (work in progress), November 2020.

[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>.

[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>.

Appendix A. Dual Mode (Group OSCORE & OSCORE)

This appendix defines the dual mode of this profile, which allows using both OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm] as security protocols, by still relying on a single Access Token.

That is, the dual mode of this profile specifies how a Client uses CoAP [RFC7252] to communicate to a single Resource Server, or CoAP over IP multicast [I-D.ietf-core-groupcomm-bis] to communicate to multiple Resource Servers that are members of a group and share a common set of resources.

In particular, the dual mode of this profile uses two complementary security protocols to provide secure communication between the Client and the Resource Server(s). That is, it defines the use of either OSCORE or Group OSCORE to protect unicast requests addressed to a single Resource Server, as well as possible responses. Additionally, it defines the use of Group OSCORE to protect multicast requests sent to a group of Resource Servers, as well as possible individual responses. Like in the main mode of this profile, the Client and the Resource Servers need to have already joined the same OSCORE group, for instance by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore], which is also based on ACE.

The Client proves its access to be authorized to the Resource Server by using an Access Token, which is bound to a key (the proof-of-possession key). This profile mode uses OSCORE to achieve proof of possession, and OSCORE or Group OSCORE to achieve server authentication.

Unlike in the main mode of this profile, where a public key is used as pop-key, this dual mode uses OSCORE-related, symmetric key material as pop-key instead. Furthermore, this dual mode provides proof of Client's membership to the correct OSCORE group, by securely binding the pre-established Group OSCORE Security Context to the pairwise OSCORE Security Context newly established between the Client and the Resource Server.

In addition to the terminology used for the main mode of this profile, the rest of this appendix refers also to "pairwise OSCORE Security Context" as to an OSCORE Security Context established between only one Client and one Resource Server, and used to communicate with OSCORE [RFC8613].

A.1. Protocol Overview

This section provides an overview on how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to secure communications between a Client and a (set of) Resource Server(s) using OSCORE [RFC8613] and/or Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Just as for main mode of this profile overviewed in Section 2, the process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] must take place before the process described in the rest of this section, and is out of the scope of this profile.

An overview of the protocol flow for the dual mode of this profile is shown in Figure 8. In the figure, it is assumed that both RS1 and RS2 are associated with the same AS. It is also assumed that C, RS1 and RS2 have previously joined an OSCORE group with Group Identifier (gid) "abcd0000", and got assigned Sender ID (sid) "0", "1" and "2" in the group, respectively. The names of messages coincide with those of [I-D.ietf-ace-oauth-authz] when applicable.

```
C                            RS1         RS2                        AS
| [--- Resource Request --->] |           |                          |
|                             |           |                          |
| [<----  AS Request ------]  |           |                          |
|       Creation Hints        |           |                          |
|                             |           |                          |
|-------- POST /token ------------------------------------------------>|
|  (aud: RS1, sid: 0, gid: abcd0000, ...) |                          |
|                             |           |                          |
|<---------------------------------- Access Token + RS Information -----|
|                             | (aud: RS1, sid: 0, gid: abcd0000, ...) |
|                             |           |                          |
|---- POST /authz-info ------>|           |                          |
|     (access_token, N1, ID1) |           |                          |
|                             |           |                          |
|<-- 2.01 Created (N2, ID2) --|           |                          |
|                             |           |                          |
/Pairwise OSCORE Sec  /Pairwise OSCORE Sec |                          |
 Context Derivation/   Context Derivation/ |                          |
|                             |           |                          |
|-------- POST /token ------------------------------------------------>|
|  (aud: RS2, sid: 0, gid: abcd0000, ...) |                          |
|                             |           |                          |
|<---------------------------------- Access Token + RS Information -----|
|                             | (aud: RS2, sid: 0, gid: abcd0000, ...) |
|                             |           |                          |
|----- POST /authz-info ------------------>|                          |
|    (access_token, N1’, ID1’) |           |                          |
|                             |           |                          |
|<-- 2.01 Created (N2’, ID2’)--------------|                          |
|                             |           |                          |
/Pairwise OSCORE Sec          |  /Pairwise OSCORE Sec                 |
 Context Derivation/          |   Context Derivation/                 |
|                             |           |                          |
|------ OSCORE Request ------>|           |                          |
|          (kid: ID2)         |           |                          |
|                             |           |                          |
|                  /proof-of-possession   |                          |
|                  Sec Context storage/   |                          |
|                             |           |                          |
|<----- OSCORE Response ------|             |                            |
|                             |             |                            |
/proof-of-possession          |             |                            |
Sec Context storage/          |             |                            |
|                             |             |                            |
|-- Group OSCORE Request -+-->|             |                            |
| (kid: 0, gid: abcd0000)  \--------------->|                            |
|                             |             |                            |
|<--- Group OSCORE Response --|             |                            |
|           (kid: 1)          |             |                            |
|                             |             |                            |
|<--- Group OSCORE Response ---------------|                            |
|           (kid: 2)          |             |                            |
|              ...            |             |                            |
```

Figure 8: Protocol Overview.

A.1.1. Pre-Conditions

The same pre-conditions for the main mode of this profile (see Section 2.1) hold for the dual mode described in this appendix.

A.1.2. Access Token Posting

After having retrieved the Access Token from the AS, the Client generates a nonce N1 and an identifier ID1 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts. The client then posts both the Access Token, N1 and its chosen 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 the dual mode of this profile, the communication with the authz-info endpoint is not protected, except for update of access rights. Note that, when using the dual mode, this request can alternatively be protected with Group OSCORE, using the Group OSCORE Security Context paired with the pairwise OSCORE Security Context originally established with the first Access Token posting.

If the Access Token is valid, the RS replies to this POST request with a 2.01 (Created) response with Content-Format = application/ace+cbor, which in a CBOR map contains a nonce N2 and an identifier ID2 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts.

A.1.3. Setup of the Pairwise OSCORE Security Context

After sending the 2.01 (Created) response, the RS sets the ID Context of the pairwise OSCORE Security Context (see Section 3 of [RFC8613]) to the Group Identifier of the OSCORE group specified in the Access Token, concatenated with N1, concatenated with N2, concatenated with the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' claim of the Access Token.

Then, the RS derives the complete pairwise OSCORE Security Context associated with the received Access Token, following Section 3.2 of [RFC8613]. In practice, the RS maintains a collection of Security Contexts with associated authorization information, for all the clients that it is currently communicating with, and the authorization information is a policy used as input when processing requests from those clients.

During the derivation process, the RS uses: the ID Context above; the nonces N1 and N2; the identifier ID1 received from the Client, set as its own OSCORE Sender ID; the identifier ID2 provided to the Client, set as its Recipient ID for the Client; and the parameters in the Access Token. The derivation process uses also the Master Secret of the OSCORE group, that the RS knows as a group member, as well as the Sender ID of the Client in the OSCORE group, which is specified in the Access Token. This ensures that the pairwise OSCORE Security Context is securely bound to the Group OSCORE Security Context of the OSCORE group.

Finally, the RS stores the association between i) the authorization information from the Access Token; and ii) the Group Identifier of the OSCORE group together with the Sender ID and the public key of the Client in that group.

After having received the nonce N2, the Client sets the ID Context in its pairwise OSCORE Security Context (see Section 3 of [RFC8613]) to the Group Identifier of the OSCORE group, concatenated with N1, concatenated with N2, concatenated with the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' parameter of the Access Token response from the AS. Then, the Client derives the complete pairwise OSCORE Security Context, following Section 3.2 of [RFC8613].

During the derivation process, the Client uses: the ID Context above, the nonces N1 and N2; the identifier ID1 provided to the RS, set as its own Recipient ID for the RS; the identifier ID2 received from the RS, set as its own OSCORE Sender ID; and the parameters received from the AS. The derivation process uses also the Master Secret of the

OSCORE group, that the Client knows as a group member, as well as its own Sender ID in the OSCORE group.

When the Client communicates with the RS using the pairwise OSCORE Security Context, the RS achieves proof-of-possession of the credentials bound to the Access Token. Also, the RS verifies that the Client is a legitimate member of the OSCORE group.

A.1.4. Secure Communication

The Client can send a request protected with OSCORE to the RS.

If the request is correctly verified, then the RS stores the pairwise OSCORE Security Context, and uses it to protect the possible response, as well as further communications with the Client, until the Access Token expires. This pairwise OSCORE Security Context is discarded when an Access Token (whether the same or different) is used to successfully derive a new pairwise OSCORE Security Context.

As discussed in Section 2 of [I-D.ietf-ace-oscore-profile], the use of random nonces N1 and N2 during the exchange between the Client and the RS prevents the reuse of AEAD nonces and keys with different messages including the possibility of two-time pads, in case of re-derivation of the pairwise OSCORE Security Context both for Clients and Resource Servers from an old non-expired Access Token, e.g. in case of reboot of either the Client or the RS.

Additionally, just as per the main mode of this profile (see Section 4.3), the Client and RS can also securely communicate by protecting messages with Group OSCORE, using the Group OSCORE Security Context already established upon joining the OSCORE group.

A.2. Client-AS Communication

This section details the Access Token POST Request that the Client sends to the /token endpoint of the AS, as well as the related Access Token response.

Section 3.2 of [RFC8613] defines how to derive a pairwise OSCORE Security Context based on a shared Master Secret and a set of other parameters, established between the OSCORE client and server, which the client receives from the AS in this exchange.

The proof-of-possession key (pop-key) received from the AS in this exchange MUST be used to build the Master Secret in OSCORE (see Appendix A.3.3 and Appendix A.3.4).

A.2.1. C-to-AS: POST to Token Endpoint

The Client-to-AS request is specified in Section 5.6.1 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.

The POST request is formatted as the analogous Client-to-AS request in the main mode of this profile (see Section 3.1), with the following modifications.

- o The parameter 'req_cnf' MUST NOT be included in the payload.

- o The parameter 'client_cred', defined in Appendix A.2.1.1 of this specification, MUST be included in the payload. This parameter includes the public key associated to the signing private key that the Client uses in the OSCORE group, whose identifier is indicated in the 'context_id' parameter.

- o The signature included in the parameter 'client_cred_verify' is computed by using the private key associated to the public key in the 'client_cred' parameter above.

An example of such a request, with payload in CBOR diagnostic notation without the tag and value abbreviations is reported in Figure 9.

```
        Header: POST (Code=0.02)
        Uri-Host: "as.example.com"
        Uri-Path: "token"
        Content-Format: "application/ace+cbor"
        Payload:
        {
          "audience" : "tempSensor4711",
          "scope" : "read",
          "context_id" : h'abcd0000',
          "salt_input" : h'00',
          "client_cred" : {
            "COSE_Key" : {
              "kty" : EC2,
              "crv" : P-256,
              "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
                      27c9e354089bbe13',
              "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
                      731e79a3b4e47120'
            }
          },
          "client_cred_verify" : h'...'
          (signature content omitted for brevity),
        }
```

Figure 9: Example C-to-AS POST /token request for an Access Token bound to a symmetric key.

Later on, the Client may want to update its current access rights, without changing the existing pairwise OSCORE Security Context with the RS. In this case, the Client MUST include in its POST request to the /token endpoint a 'req_cnf' parameter, defined in Section 3.1 of [I-D.ietf-ace-oauth-params], which MUST include a 'kid' field, as defined in Section 3.1 of [RFC8747]. The 'kid' field has as value a CBOR byte string containing the OSCORE_Input_Material Identifier (assigned as discussed in Appendix A.2.2).

This identifier, together with other information such as audience, can be used by the AS to determine the shared secret bound to the proof-of-possession Access 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 5.6.3 of [I-D.ietf-ace-oauth-authz].

This POST request for updating the rights of an Access Token MUST NOT include the parameters 'salt_input', 'context_id', 'client_cred' and 'client_cred_verify'.

An example of such a request, with payload in CBOR diagnostic notation without the tag and value abbreviations is reported in Figure 10.

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

Figure 10: Example C-to-AS POST /token request for updating rights to an Access Token bound to a symmetric key.

A.2.1.1. 'client_cred' Parameter

The 'client_cred' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.6.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides an asymmetric key that the Client wishes to use as its own public key, but which is not used as proof-of-possession key.

This parameter follows the syntax of the 'cnf' claim from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. Alternative Value Types defined in future specifications are fine to consider if indicating a non-encrypted asymmetric key.

A.2.2. AS-to-C: Access Token

After having verified 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 MUST verify the signature in the 'client_cred_verify' parameter, by using the public key specified in the 'client_cred' parameter. If the verification fails, the AS considers the Client request invalid. The AS does not perform this operation when asked to update a previously released Access Token.

If all verifications are successful, the AS responds as defined in Section 5.6.2 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 5.6.3 of [I-D.ietf-ace-oauth-authz].

The AS can signal that the use of OSCORE and Group OSCORE is REQUIRED for a specific Access Token by including the 'profile' parameter with the value "coap_group_oscore" in the Access Token response. This means that the Client MUST use OSCORE and/or Group OSCORE towards all the Resource Servers for which this Access Token is valid.

In particular, the Client MUST follow Appendix A.3.3 to derive the pairwise OSCORE Security Context to use for communications with the RS. Instead, the Client has already established the related Group OSCORE Security Context to communicate with members of the OSCORE group, upon previously joining that group.

Usually, it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile negotiation can be omitted.

In contrast with the main mode of this profile, the Access Token response to the Client is analogous to the one in the OSCORE profile of ACE, as described in Section 3.2 of [I-D.ietf-ace-oscore-profile]. In particular, the AS provides an OSCORE_Input_Material object, which is defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile] and included in the 'cnf' parameter (see Section 3.2 of [I-D.ietf-ace-oauth-params]) of the Access Token response.

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.

In the issued Access Token, the AS MUST include as metadata the same information as defined in the main mode of this profile (see Section 3.2) with the following modifications.

o The public key that the client uses in the OSCORE group and specified in the 'client_cred' parameter of the Token request (see Appendix A.2.1) MUST also be included in the Access Token. If the Access Token is a CWT, the AS MUST include it in the 'client_cred' claim of the Access Token, defined in Appendix A.2.2.2 of this specification.

o The OSCORE_Input_Material specified in the 'cnf' parameter of the Access Token response MUST also be included in the Access Token. If the Access Token is a CWT, the same OSCORE_Input_Material included in the 'cnf' parameter of the Access Token response MUST be included in the 'osc' field (see Section 9.5 of [I-D.ietf-ace-oscore-profile]) of the 'cnf' claim of the Access Token.

Figure 11 shows an example of such an AS response, with payload in CBOR diagnostic notation without the tag and value abbreviations.

```
     Header: Created (Code=2.01)
     Content-Type: "application/ace+cbor"
     Payload:
     {
       "access_token" : h'8343a1010aa2044c53 ...'
        (remainder of CWT omitted for brevity),
       "profile" : "coap_group_oscore",
       "expires_in" : 3600,
       "cnf" : {
         "osc" : {
           "alg" : "AES-CCM-16-64-128",
           "id"  : h'01',
           "ms" : h'f9af838368e353e78888e1426bd94e6f',
           "salt" : h'1122',
           "contextId" : h'99'
         }
       }
     }
```

Figure 11: Example AS-to-C Access Token response with the Group OSCORE profile.

Figure 12 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```
        {
          "aud" : "tempSensorInLivingRoom",
          "iat" : "1360189224",
          "exp" : "1360289224",
          "scope" :  "temperature_g firmware_p",
          "cnf" : {
            "osc" : {
              "alg" : "AES-CCM-16-64-128",
              "id"  : h'01',
              "ms" : h'f9af838368e353e78888e1426bd94e6f',
              "salt" : h'1122',
              "contextId" : h'99'
          },
          "salt_input" : h'00',
          "contextId_input" : h'abcd0000',
          "client_cred" : {
            "COSE_Key" : {
              "kty" : EC2,
              "crv" : P-256,
              "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
                      27c9e354089bbe13',
              "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
                      731e79a3b4e47120'
            }
          }
        }
```

Figure 12: Example CWT with OSCORE parameters (CBOR diagnostic notation).

The same CWT as in Figure 12 and encoded in CBOR is shown in Figure 13, using the value abbreviations defined in [I-D.ietf-ace-oauth-authz] and [RFC8747].

NOTE: 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.

```
   A8                                   # map(8)
      03                                # unsigned(3)
      76                                # text(22)
         74656D7053656E736F72496E4C6976696E67526F6F6D
      06                                # unsigned(6)
      1A 5112D728                       # unsigned(1360189224)
      04                                # unsigned(4)
      1A 51145DC8                       # unsigned(1360289224)
      09                                # unsigned(9)
      78 18                             # text(24)
```

```
         74656D70657261747572655F67206669726D776172655F70
      08                                   # unsigned(8)
      A1                                   # map(1)
         04                                # unsigned(4)
         A5                                # map(5)
            04                             # unsigned(4)
            0A                             # unsigned(10)
            02                             # unsigned(2)
            41                             # bytes(1)
               01                          # "\x01"
            01                             # unsigned(1)
            50                             # bytes(16)
               F9AF838368E353E78888E1426BD94E6F
            05                             # unsigned(5)
            42                             # bytes(2)
               1122                        # "\x11\""
            06                             # unsigned(6)
            41                             # bytes(1)
               99                          # "\x99"
      18 3C                                # unsigned(60)
      41                                   # bytes(1)
         00
      18 3D                                # unsigned(61)
      44                                   # bytes(4)
         ABCD0000
      18 3E                                # unsigned(62)
      A1                                   # map(1)
         01                                # unsigned(1)
         A4                                # map(4)
            01                             # unsigned(1)
            02                             # unsigned(2)
            20                             # negative(0)
            01                             # unsigned(1)
            21                             # negative(1)
            58 20                          # bytes(32)
               D7CC072DE2205BDC1537A543D53C60A6ACB62ECCD890C7FA27C9
               E354089BBE13
            22                             # negative(2)
            58 20                          # bytes(32)
               F95E1D4B851A2CC80FFF87D8E23F22AFB725D535E515D020731E
               79A3B4E47120
```

Figure 13: Example CWT with OSCORE parameters.

If the Client has requested an update to its access rights using the same pairwise OSCORE Security Context, which is valid and authorized, the AS MUST omit the 'cnf' parameter in the response to the client.

Instead, the updated Access Token conveyed in the AS-to-C response MUST include a 'cnf' claim specifying a 'kid' field, as defined in Section 3.1 of [RFC8747]. The response from the AS MUST carry the OSCORE Input Material identifier in the 'kid' field within the 'cnf' claim of the Access Token. That is, the 'kid' field is a CBOR byte string, with value the same value of the 'kid' field of the 'req_cnf' parameter from the C-to-AS request for updating rights to the Access Token (see Figure 10). This information needs to be included in the Access Token, in order for the RS to identify the previously generated pairwise OSCORE Security Context.

Figure 14 shows an example of such an AS response, with payload in CBOR diagnostic notation without the tag and value abbreviations. 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 CWT omitted for brevity),
        "profile" : "coap_group_oscore",
        "expires_in" : 3600
      }
```

Figure 14: Example AS-to-C Access Token response with the Group OSCORE profile, for update of access rights.

Figure 15 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim for update of access rights, in CBOR diagnostic notation without tag and value abbreviations.

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

Figure 15: Example CWT with OSCORE parameters for update of access rights.

A.2.2.1. Public Key Hash as Client Credential

As a possible optimization to limit the size of the Access Token, the AS may specify as value of the 'client_cred' claim simply the hash of the Client's public key. The specifically used hash-function MUST be collision-resistant on byte-strings, and MUST be selected from the "Named Information Hash Algorithm" Registry defined in Section 9.4 of [RFC6920].

In particular, the AS provides the Client with an Access Token as defined in Appendix A.2.2, with the following differences.

The AS prepares INPUT_HASH as follows, with | denoting byte string concatenation.

- o If the public key has COSE Key Type OKP, INPUT_HASH = i, where 'i' is the x-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, encoded as a CBOR byte string.
- o If the public key has COSE Key Type EC2, INPUT_HASH = (i_1 | i_2), where 'i_1' and 'i_2' are the x-parameter and y-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, respectively, each encoded as a CBOR byte string.
- o If the public key has COSE Key Type RSA, INPUT_HASH = (i_1 | i_2), where 'i_1' and 'i_2' are the n-parameter and e-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, respectively, each encoded as a CBOR byte string.

Then, the AS hashes INPUT_HASH according to the procedure described in [RFC6920], with the output OUTPUT_HASH in binary format, as described in Section 6 of [RFC6920].

Finally, the AS includes a single entry within the 'client_cred' claim of the Access Token. This entry has label "kid" (3) defined in Section 3.1 of [RFC8747], and value a CBOR byte string wrapping OUTPUT_HASH.

Upon receiving the Access Token, the RS processes it according to Appendix A.3.2, with the following differences.

The RS considers the content of the 'client_cred' claim as the hash of the public key associated to the signing private key that the Client uses in the OSCORE group, which is identified by the 'context_id' parameter.

The RS MAY additionally request the Group Manager of the OSCORE group for the public key of that Client, as described in

[I-D.ietf-ace-key-groupcomm-oscore], specifying as Sender ID of that Client in the OSCORE group the value of the 'salt_input' claim included in the Access Token.

In such a case, the RS MUST check that the hash of the key retrieved from the Group Manager matches the hash retrieved from the 'client_cred' claim of the Access Token. The RS MUST calculate the hash using the same method as the AS described above, and using the same hash function. The hash function used can be determined from the information conveyed in the 'client_cred' claim, as the procedure described in [RFC6920] also encodes the used hash function as metadata of the hash value.

A.2.2.2. Client Credential Claim

The 'client_cred' claim provides an asymmetric key that the Client owning the Access Token wishes to use as its own public key, but which is not used as proof-of-possession key.

This parameter follows the syntax of the 'cnf' claim from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. Alternative Value Types defined in future specifications are fine to consider if indicating a non-encrypted asymmetric key.

A.3. Client-RS Communication

This section details the POST request and response to the /authz-info endpoint between the Client and the RS. With respect to the exchanged messages and their content, the Client and the RS perform as defined in Section 4 of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

That is, the Client generates a nonce N1 and posts it to the RS, together with: an identifier ID1 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts; and the Access Token that includes the material provisioned by the AS.

Then, the RS generates a nonce N2, and an identifier ID2 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts. After that, the RS derives a pairwise OSCORE Security Context as described in Section 3.2 of [RFC8613]. In particular, it uses the two nonces and the two identifiers established with the Client, as well as two shared secrets together with additional pieces of information specified in the Access Token.

Both the client and the RS generate the pairwise OSCORE Security Context using the pop-key as part of the OSCORE Master Secret. In addition, the derivation of the pairwise OSCORE Security Context takes as input also information related to the OSCORE group, i.e. the Master Secret and Group Identifier of the group, as well as the Sender ID of the Client in the group. Hence, the derived pairwise OSCORE Security Context is also securely bound to the Group OSCORE Security Context of the OSCORE Group. Thus, the proof-of-possession required to bind the Access Token to the Client occurs after the first OSCORE message exchange.

Therefore, an attacker using a stolen Access Token cannot generate a valid pairwise OSCORE Security Context and thus cannot prove possession of the pop-key. Also, if a Client legitimately owns an Access Token but has not joined the OSCORE group, that Client cannot generate a valid pairwise OSCORE Security Context either, since it lacks the Master Secret used in the OSCORE group.

Besides, just as in the main mode (see Section 4), the RS is able to verify whether the Client has indeed the claimed Sender ID and public key in the OSCORE group.

A.3.1. C-to-RS POST to authz-info Endpoint

The Client MUST generate a nonce N1, an OSCORE Recipient ID (ID1), and post them to the /authz-info endpoint of the RS together with the Access Token, as defined in Section 4.1 of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

The same recommendations, considerations and behaviors defined in Section 4.1 of [I-D.ietf-ace-oscore-profile] hold.

If the Client has already posted a valid Access Token, has already established a pairwise OSCORE Security Context with the RS, and wants to update its access rights, the Client can do so by posting the new Access Token (retrieved from the AS and specifying the updated set of access rights) to the /authz-info endpoint.

The Client MUST protect the request using either the pairwise OSCORE Security Context established during the first Access Token exchange, or the Group OSCORE Security Context associated to that pairwise OSCORE Security Context.

In either case, the Client MUST only send the Access Token in the payload, i.e. no nonce or identifier are sent. After proper verification (see Section 4.2 of [I-D.ietf-ace-oscore-profile]), the RS will replace the old Access Token with the new one, maintaining the same pairwise OSCORE Security Context and Group OSCORE Security Context.

A.3.2. RS-to-C: 2.01 (Created)

The RS MUST verify the validity of the Access Token as defined in Section 4.2, with the following modifications.

- o The RS checks that the 'cnf' claim is included in the Access Token and that it contains an OSCORE_Input_Material object.

- o The RS checks that the 'client_cred' claim is included in the Access Token.

- o The RS considers the content of the 'client_cred' claim as the public key associated to the signing private key of the Client in the OSCORE group, whose GID is specified in the 'contextId_input' claim. The RS can compare this public key with the public key of the claimed Client retrieved from the Group Manager (see Section 4.2).

If any of the checks fails, the RS MUST consider the Access Token non valid, and MUST respond to the Client with an error response code equivalent to the CoAP code 4.00 (Bad Request).

If the Access Token is valid and further checks on its content are successful, the RS MUST generate a nonce N2, an OSCORE Recipient ID (ID2), and include them in the 2.01 (Created) response to the Client, as defined in Section 4.2 of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

Further recommendations, considerations and behaviors defined in Section 4.2 of [I-D.ietf-ace-oscore-profile] hold for this specification.

A.3.3. OSCORE Setup - Client Side

Once having received the 2.01 (Created) response from the RS, following the POST request to the authz-info endpoint, the Client MUST extract the nonce N2 from the 'nonce2' parameter, and the Client identifier from the 'ace_server_recipientid' parameter in the CBOR map of the response payload. Note that this identifier is used by C as Sender ID in the pairwise OSCORE Security Context to be established with the RS, and is different as well as unrelated to the Sender ID of C in the OSCORE group.

Then, the Client performs the following actions, in order to set up and fully derive the pairwise OSCORE Security Context for communicating with the RS.

o The Client MUST set the ID Context of the pairwise OSCORE Security Context as the concatenation of: i) GID, i.e. the Group Identifier of the OSCORE group, as specified by the Client in the 'context_id' parameter of the Client-to-AS request; ii) the nonce N1; iii) the nonce N2; and iv) CID, i.e. the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' parameter of the Access Token response from the AS. The concatenation occurs in this order: ID Context = GID | N1 | N2 | CID, where | denotes byte string concatenation.

o The Client MUST set the updated Master Salt of the pairwise OSCORE Security Context as the concatenation of SaltInput, MSalt, the nonce N1, the nonce N2 and GMSalt, where: i) SaltInput is the Sender ID that the Client has in the OSCORE group, which is known to the Client as a member of the OSCORE group; ii) MSalt is the (optional) Master Salt in the pairwise OSCORE Security Context (received from the AS in the Token); and iii) GMSalt is the (optional) Master Salt in the Group OSCORE Security Context, which is known to the Client as a member of the OSCORE group. The concatenation occurs in this order: Master Salt = SaltInput | MSalt | N1 | N2 | GMSalt, where | denotes byte string concatenation. Optional values, if not specified, are not included in the concatenation. The five parameters SaltInput, MSalt, N1, N2 and GMSalt are to be concatenated as encoded CBOR byte strings. An example of Master Salt construction using CBOR encoding is given in Figure 16.

```
   SaltInput, MSalt, N1, N2 and GMSalt, in CBOR diagnostic notation:
         SaltInput = h'00'
         MSalt = h'f9af838368e353e78888e1426bd94e6f'
         N1 = h'018a278f7faab55a'
         N2 = h'25a8991cd700ac01'
         GMSalt = h'99'

   SaltInput, MSalt, N1, N2 and GMSalt, as CBOR encoded byte strings:
         SaltInput = 0x4100
         MSalt = 0x50f9af838368e353e78888e1426bd94e6f
         N1 = 0x48018a278f7faab55a
         N2 = 0x4825a8991cd700ac01
         GMSalt = 0x4199

   Master Salt = 0x41 00
                   50 f9af838368e353e78888e1426bd94e6f
                   48 018a278f7faab55a
                   48 25a8991cd700ac01
                   41 99
```

Figure 16: Example of Master Salt construction using CBOR encoding.

- The Client MUST set the Master Secret of the pairwise OSCORE Security Context to the concatenation of MSec and GMSec, where: i) MSec is the value of the 'ms' parameter in the OSCORE_Input_Material of the 'cnf' parameter, received from the AS in Appendix A.2.2; while ii) GMSec is the Master Secret of the Group OSCORE Security Context, which is known to the Client as a member of the OSCORE group.

- The Client MUST set the Recipient ID as ace_client_recipientid, sent as described in Appendix A.3.1.

- The Client MUST set the Sender ID as ace_server_recipientid, received as described in Appendix A.3.1.

- The Client MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version as indicated in the corresponding parameters received from the AS in Appendix A.2.2, if present in the OSCORE_Input_Material of the 'cnf' parameter. In case these parameters are omitted, the default values SHALL be used as described in Section 3.2 and 5.4 of [RFC8613].

Finally, the client MUST derive the complete pairwise OSCORE Security Context following Section 3.2.1 of [RFC8613].

From then on, when communicating with the RS to access the resources as specified by the authorization information, the Client MUST use the newly established pairwise OSCORE Security Context or the Group OSCORE Security Context of the OSCORE Group where both the Client and the RS are members.

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, then the client MUST stop the exchange, and MUST NOT derive the pairwise OSCORE Security Context. The Client MAY restart the exchange, to get the correct security material.

The Client can use this pairwise OSCORE Security Context to send requests to the RS protected with OSCORE. Besides, the Client can use the Group OSCORE Security Context for protecting unicast requests to the RS, or multicast requests to the OSCORE group including also the RS.

Note that the ID Context of the pairwise OSCORE Security 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, the 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 pairwise OSCORE Security Context will not match the "contextId" parameter of the corresponding OSCORE_Input_Material. Running the procedure in Appendix B.2 of [RFC8613] results in the keying material in the pairwise OSCORE Security Contexts of the Client and RS being updated. The Client can achieve the same result by re-posting the Access Token as described in Section 4.1 of [I-D.ietf-ace-oscore-profile], although without updating the ID Context.

A.3.4. OSCORE Setup - Resource Server Side

After validation of the Access Token as defined in Appendix A.3.2 and after sending the 2.01 (Created) response, the RS performs the following actions, in order to set up and fully derive the pairwise OSCORE Security Context created to communicate with the Client.

- o The RS MUST set the ID Context of the pairwise OSCORE Security Context as the concatenation of: i) GID, i.e. the Group Identifier of the OSCORE group, as specified in the 'contextId' parameter of the OSCORE_Input_Material, in the 'cnf' claim of the Access Token received from the Client (see Appendix A.3.1); ii) the nonce N1; iii) the nonce N2; and iv) CID which is the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' claim of the Access Token. The concatenation occurs in this order: ID Context = GID | N1 | N2 | CID, where | denotes byte string concatenation.

- o The RS MUST set the new Master Salt of the pairwise OSCORE Security Context as the concatenation of SaltInput, MSalt, the nonce N1, the nonce N2 and GMSalt, where: i) SaltInput is the Sender ID that the Client has in the OSCORE group, as specified in the 'salt_input' claim included in the Access Token received from the Client (see Appendix A.3.1); ii) MSalt is the (optional) Master Salt in the pairwise OSCORE Security Context as specified in the 'salt' parameter in the OSCORE_Input_Material of the 'cnf' claim, included in the Access Token received from the Client; and iii) GMSalt is the (optional) Master Salt in the Group OSCORE Security Context, which is known to the RS as a member of the OSCORE group. The concatenation occurs in this order: Master Salt = SaltInput | MSalt | N1 | N2 | GMSalt, where | denotes byte string concatenation. Optional values, if not specified, are not included in the concatenation. The same considerations for building the Master Salt, considering the inputs as encoded CBOR byte strings as in Figure 16, hold also for the RS.

- o The RS MUST set the Master Secret of the pairwise OSCORE Security Context to the concatenation of MSec and GMSec, where: i) MSec is the value of the 'ms' parameter in the OSCORE_Input_Material of the 'cnf' claim, included in the Access Token received from the Client (see Appendix A.3.1); while ii) GMSec is the Master Secret of the Group OSCORE Security Context, which is known to the RS as a member of the OSCORE group.

- o The RS MUST set the Recipient ID as ace_server_recipientid, sent as described in Appendix A.3.2.
- o The RS MUST set the Sender ID as ace_client_recipientid, received as described in Appendix A.3.2.
- o The RS MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version from the corresponding parameters received from the Client in the Access Token (see Appendix A.3.1), if present in the OSCORE_Input_Material of the 'cnf' claim. In case these parameters are omitted, the default values SHALL be used as described in Section 3.2 and 5.4 of [RFC8613].

Finally, the RS MUST derive the complete pairwise OSCORE Security Context following Section 3.2.1 of [RFC8613].

Once having completed the derivation above, the RS MUST associate the authorization information from the Access Token with the just established pairwise OSCORE Security Context. Furthermore, as defined in Section 4.2, the RS MUST associate the authorization information from the Access Token with the Group OSCORE Security Context.

Then, the RS uses this pairwise OSCORE Security Context to verify requests from and send responses to the Client protected with OSCORE, when this Security Context is used. If OSCORE verification fails, error responses are used, as specified in Section 8 of [RFC8613]. Besides, the RS uses the Group OSCORE Security Context to verify (multicast) requests from and send responses to the Client protected with Group OSCORE. If Group OSCORE verification fails, error responses are used, as specified in Section 8 and Section 9 of [I-D.ietf-core-oscore-groupcomm]. Additionally, for every incoming request, if OSCORE or Group OSCORE verification succeeds, the verification of access rights is performed as described in Appendix A.3.5.

After the expiration of the Access Token related to a pairwise OSCORE Security Context and to a Group OSCORE Security Context, the RS MUST NOT use the pairwise OSCORE Security Context and MUST respond with an unprotected 4.01 (Unauthorized) error message to received requests that correspond to a security context with an expired Access Token. Also, if the Client uses the Group OSCORE Security Context to send a request for any resource intended for OSCORE group members and that requires an active Access Token, the RS MUST respond with a 4.01 (Unauthorized) error message protected with the Group OSCORE Security Context.

The same considerations, related to the value of the ID Context changing, as in Appendix A.3.3 hold also for the RS.

A.3.5. Access Rights Verification

The RS MUST follow the procedures defined in Section 4.4.

Additionally, if the RS receives an OSCORE-protected request from a Client, the RS processes it according to [RFC8613].

If the OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the authorization information from the Access Token associated to the pairwise OSCORE Security Context and to the Group OSCORE Security Context. Then, the RS MUST verify that the action requested on the resource is authorized.

The response code MUST be 4.01 (Unauthorized) if the RS has no valid Access Token for the Client.

A.4. Secure Communication with the AS

The same considerations for secure communication with the AS as defined in Section 5 hold.

A.5. Discarding the Security Context

The Client and the RS MUST follow what is defined in Section 6 of [I-D.ietf-ace-oscore-profile] about discarding the pairwise OSCORE Security Context.

Additionally, they MUST follow what is defined in the main mode of the profile (see Section 6), with respect to the Group OSCORE Security Context.

The Client or RS can acquire a new Group OSCORE Security Context, by re-joining the OSCORE group, e.g. by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. In such a case, the Client SHOULD request a new Access Token and post it to the RS, in order to establish a new pairwise OSCORE Security Context and bind it to the Group OSCORE Security Context obtained upon re-joining the group.

A.6. CBOR Mappings

The new parameters defined in this document MUST be mapped to CBOR types as specified in Figure 6, with the following addition, using the given integer abbreviation for the map key.

Parameter name	CBOR Key	Value Type
client_cred	TBD6	map

Figure 17: CBOR mappings for new parameters.

The new claims defined in this document MUST be mapped to CBOR types as specified in Figure 7, with the following addition, using the given integer abbreviation for the map key.

Claim name	CBOR Key	Value Type
client_cred	TBD7	map

Figure 18: CBOR mappings for new claims.

A.7. Security Considerations

The dual mode of this profile inherits the security considerations from the main mode (see Section 8), as well as from the security considerations of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile]. Also, the security considerations about OSCORE [RFC8613] hold for the dual mode of this profile, as to the specific use of OSCORE.

A.8. Privacy Considerations

The same privacy considerations as defined in the main mode of this profile apply (see Section 9).

In addition, as this profile mode also uses OSCORE, the privacy considerations from [RFC8613] apply as well, as to the specific use of OSCORE.

Furthermore, this profile mode inherits the privacy considerations from the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

Appendix B. Profile Requirements

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

- o (Optional) discovery process of how the Client finds the right AS for an RS it wants to send a request to: Not specified.

- o Communication protocol the Client and the RS must use: CoAP.

- o Security protocol(s) the Client and RS must use: Group OSCORE, i.e. exchange of secure messages by using a pre-established Group OSCORE Security Context. The optional dual mode defined in Appendix A additionally uses OSCORE, i.e. establishment of a pairwise OSCORE Security Context and exchange of secure messages.

- o How the Client and the RS mutually authenticate: Explicitly, by possession of a common Group OSCORE Security Context, and by either: usage of digital counter signatures embedded in messages, if protected with the group mode of Group OSCORE; or protection of messages with the pairwise mode of Group OSCORE, by using pairwise symmetric keys, derived from the asymmetric keys of the two peers exchanging the message. Note that the mutual authentication is not completed before the Client has verified an OSCORE or a Group OSCORE response using the corresponding security context.

- o Content-format of the protocol messages: "application/ace+cbor".

- o Proof-of-Possession protocol(s) and how to select one; which key types (e.g. symmetric/asymmetric) supported: Group OSCORE algorithms; distributed and verified asymmetric keys. In the optional dual mode defined in Appendix A: OSCORE algorithms; pre-established symmetric keys.

- o profile identifier: coap_group_oscore

- o (Optional) how the RS talks to the AS for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE).

- o How the client talks to the AS for requesting a token: HTTP/CoAP (+ TLS/DTLS/OSCORE).

- o How/if the authz-info endpoint is protected: Not protected.

- o (Optional) other methods of token transport than the authz-info endpoint: Not specified.

Acknowledgments

The authors sincerely thank Benjamin Kaduk, John Mattsson, Dave Robin, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Ludwig Seitz
Combitech
Djaeknegatan 31
Malmoe SE-21135 Malmoe
Sweden

Email: ludwig.seitz@combitech.se

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

M. Tiloca
R. Hoeglund
RISE AB
P. van der Stok
Consultant
F. Palombini
K. Hartke
Ericsson AB
July 13, 2020

Admin Interface for the OSCORE Group Manager
draft-tiloca-ace-oscore-gm-admin-02

Abstract

Group communication for CoAP can be secured using Group Object Security for Constrained RESTful Environments (Group OSCORE). A Group Manager is responsible to handle the joining of new group members, as well as to manage and distribute the group key material. This document defines a RESTful admin interface at the Group Manager, that allows an Administrator entity to create and delete OSCORE groups, as well as to retrieve and update their configuration. The ACE framework for Authentication and Authorization is used to enforce authentication and authorization of the Administrator at the Group Manager. Protocol-specific transport profiles of ACE are used to achieve communication security, proof-of-possession and server authentication.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction   3
  1.1. Terminology   4
2. Group Administration   5
  2.1. Getting Access to the Group Manager   6
  2.2. Managing OSCORE Groups   7
  2.3. Group Configurations   8
    2.3.1. Group Configuration Representation   8
    2.3.2. Default Values 10
  2.4. Discovery 10
  2.5. Collection Representation 10
  2.6. Interactions 11
    2.6.1. Get All Groups Configurations 11
    2.6.2. Fetch Group Configurations By Filters 12
    2.6.3. Create a New Group Configuration 13
    2.6.4. Retrieve a Group Configuration 17
    2.6.5. Update a Group Configuration 18
    2.6.6. Delete a Group Configuration 21
3. Security Considerations 23
4. IANA Considerations 23
  4.1. ACE Groupcomm Parameters Registry 23
  4.2. Resource Types 24
5. References 24
  5.1. Normative References 25
  5.2. Informative References 27
Acknowledgments 28
Authors' Addresses 28

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] can be used in group communication environments where messages are also exchanged over IP multicast [I-D.dijk-core-groupcomm-bis]. Applications relying on CoAP can achieve end-to-end security at the application layer by using Object Security for Constrained RESTful Environments (OSCORE) [RFC8613], and especially Group OSCORE [I-D.ietf-core-oscore-groupcomm] in group communication scenarios.

When group communication for CoAP is protected with Group OSCORE, nodes are required to explicitly join the correct OSCORE group. To this end, a joining node interacts with a Group Manager (GM) entity responsible for that group, and retrieves the required key material to securely communicate with other group members using Group OSCORE.

The method in [I-D.ietf-ace-key-groupcomm-oscore] specifies how nodes can join an OSCORE group through the respective Group Manager. Such a method builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], so ensuring a secure joining process as well as authentication and authorization of joining nodes (clients) at the Group Manager (resource server).

In some deployments, the application running on the Group Manager may know when a new OSCORE group has to be created, as well as how it should be configured and later on updated or deleted, e.g. based on the current application state or on pre-installed policies. In this case, the Group Manager application can create and configure OSCORE groups when needed, by using a local application interface. However, this requires the Group Manager to be application-specific, which in turn leads to error prone deployments and is poorly flexible.

In other deployments, a separate Administrator entity, such as a Commissioning Tool, is directly responsible for creating and configuring the OSCORE groups at a Group Manager, as well as for maintaining them during their whole lifetime until their deletion. This allows the Group Manager to be agnostic of the specific applications using secure group communication.

This document specifies a RESTful admin interface at the Group Manager, intended for an Administrator, as a separate entity external to the Group Manager and its application. The interface allows the Administrator to create and delete OSCORE groups, as well as to configure and update their configuration.

Interaction examples are provided, in Link Format [RFC6690] and CBOR [RFC7049], as well as in CoRAL [I-D.ietf-core-coral]. While all the CoRAL examples use the CoRAL textual serialization format, the CBOR or JSON [RFC8259] binary serialization format is used when sending such messages on the wire.

The ACE framework is used to ensure authentication and authorization of the Administrator (client) at the Group Manager (resource server). In order to achieve communication security, proof-of-possession and server authentication, the Administrator and the Group Manager leverage protocol-specific transport profiles of ACE, such as [I-D.ietf-ace-oscore-profile][I-D.ietf-ace-dtls-authorize]. These include also possible forthcoming transport profiles that comply with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

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 related to CBOR [RFC7049] and COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], the CoAP protocol [RFC7252], as well as the protection and processing of CoAP messages using OSCORE [RFC8613], also in group communication scenarios using Group OSCORE [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications among members are secured using Group OSCORE.

Readers are also expected to be familiar with the terms and concept related to the management of keying material for groups in ACE defined in [I-D.ietf-ace-key-groupcomm], and in particular to the joining process for OSCORE groups defined in [I-D.ietf-ace-key-groupcomm-oscore]. These include the concept of group-membership resource hosted by the Group Manager, that new members access to join the OSCORE group, while current members can access to retrieve updated keying material.

Readers are also expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

This document also refers to the following terminology.

- Administrator: entity responsible to create, configure and delete OSCORE groups at a Group Manager.

- Group name: stable and invariant name of an OSCORE group. The group name MUST be unique under the same Group Manager, and MUST include only characters that are valid for a URI path segment, namely unreserved and pct-encoded characters [RFC3986].

- Group-collection resource: a single-instance resource hosted by the Group Manager. An Administrator accesses a group-collection resource to create a new OSCORE group, or to retrieve the list of existing OSCORE groups, under that Group Manager. As an example, this document uses /manage as the url-path of the group-collection resource; implementations are not required to use this name, and can define their own instead.

- Group-configuration resource: a resource hosted by the Group Manager, associated to an OSCORE group under that Group Manager. A group-configuration resource is identifiable with the invariant group name of the respective group. An Administrator accesses a group-configuration resource to retrieve or update the configuration of the respective OSCORE group, or to delete that group. The url-path to a group-configuration resource has NAME as last segment, with NAME the invariant group name assigned upon its creation. Building on the considered url-path of the group-collection resource, this document uses /manage/NAME as the url-path of a group-configuration resource; implementations are not required to use this name, and can define their own instead.

- Admin endpoint: an endpoint at the Group Manager associated to the group-collection resource or to a group-configuration resource hosted by that Group Manager.

2. Group Administration

With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- The Group Manager acts as Resource Server (RS). It provides one single group-collection resource, and one group-configuration resource per existing OSCORE group. Each of those is exported by a distinct admin endpoint.

- The Administrator acts as Client (C), and requests to access the group-collection resource and group-configuration resources, by accessing the respective admin endpoint at the Group Manager.

- The Authorization Server (AS) authorizes the Administrator to access the group-collection resource and group-configuration resources at a Group Manager. Multiple Group Managers can be associated to the same AS. The AS MAY release Access Tokens to the Administrator for other purposes than accessing admin endpoints of registered Group Managers.

2.1. Getting Access to the Group Manager

All communications between the involved entities rely on the CoAP protocol and MUST be secured.

In particular, communications between the Administrator and the Group Manager leverage protocol-specific transport profiles of ACE to achieve communication security, proof-of-possession and server authentication. To this end, the AS may explicitly signal the specific transport profile to use, consistently with requirements and assumptions defined in the ACE framework [I-D.ietf-ace-oauth-authz].

With reference to the AS, communications between the Administrator and the AS (/token endpoint) as well as between the Group Manager and the AS (/introspect endpoint) can be secured by different means, for instance using DTLS [RFC6347][I-D.ietf-tls-dtls13] or OSCORE [RFC8613]. Further details on how the AS secures communications (with the Administrator and the Group Manager) depend on the specifically used transport profile of ACE, and are out of the scope of this specification.

In order to get access to the Group Manager for managing OSCORE groups, an Administrator performs the following steps.

1. The Administrator requests an Access Token from the AS, in order to access the group-collection and group-configuration resources on the Group Manager. The Administrator will start or continue using secure communications with the Group Manager, according to the response from the AS.

2. The Administrator transfers authentication and authorization information to the Group Manager by posting the obtained Access Token, according to the used profile of ACE, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile]. After that, the Administrator must have secure communication established with the Group Manager, before performing any admin operation on that Group Manager. Possible ways to provide secure

communication are DTLS [RFC6347][I-D.ietf-tls-dtls13] and OSCORE [RFC8613]. The Administrator and the Group Manager maintain the secure association, to support possible future communications.

3. The Administrator performs admin operations at the Group Manager, as described in the following sections. These include the retrieval of the existing OSCORE groups, the creation of new OSCORE groups, the update and retrieval of group configurations, and the removal of OSCORE groups. Messages exchanged among the Administrator and the Group Manager are specified in Section 2.6.

2.2. Managing OSCORE Groups

Figure 1 shows the resources of a Group Manager available to an Administrator.

```
                   ___
         Group    /   \
      Collection  \___/
                       \
                        \____________________
                         \___    \___        \___
                         /   \   /   \  ...  /   \         Group
                         \___/   \___/       \___/   Configurations
```

Figure 1: Resources of a Group Manager

The Group Manager exports a single group-collection resource. The full interface for the group-collection resource allows the Administrator to:

- o Retrieve the list of existing OSCORE groups, possibly by filters.
- o Create a new OSCORE group, specifying its invariant group name and, optionally, its configuration.

The Group Manager exports one group-configuration resource for each of its OSCORE groups. Each group-configuration resource is identified by the group name specified upon creating the group. The full interface for a group-configuration resource allows the Administrator to:

- o Retrieve the current configuration of the OSCORE group.
- o Update the current configuration of the OSCORE group.
- o Delete the OSCORE group.

2.3. Group Configurations

A group configuration consists of a set of parameters.

2.3.1. Group Configuration Representation

The group configuration representation is a CBOR map which MUST include configuration properties and status properties.

2.3.1.1. Configuration Properties

The CBOR map MUST include the following configuration parameters:

o ’hkdf’, defined in Section 4.1 of this document, specifies the HKDF algorithm used in the OSCORE group, encoded as a CBOR text string. Possible values are the same ones admitted for the ’hkdf’ parameter of the "OSCORE Security Context Parameters" registry, defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile].

o ’alg’, defined in Section 4.1 of this document, specifies the AEAD algorithm used in the OSCORE group, encoded as a CBOR text string. Possible values are the same ones admitted for the ’alg’ parameter of the "OSCORE Security Context Parameters" registry, defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile].

o ’cs_alg’, defined in Section 4.1 of this document, specifies the countersignature algorithm used in the OSCORE group, encoded as a CBOR text string or integer. Possible values are the same ones admitted for the ’cs_alg’ parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

o ’cs_params’, defined in Section 4.1 of this document, specifies the additional parameters for the countersignature algorithm used in the OSCORE group, encoded as a CBOR array. Possible formats and values are the same ones admitted for the ’cs_params’ parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

o ’cs_key_params’, defined in Section 4.1 of this document, specifies the additional parameters for the key used with the countersignature algorithm in the OSCORE group, encoded as a CBOR array. Possible formats and values are the same ones admitted for the ’cs_key_params’ parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

- 'cs_key_enc', defined in Section 4.1 of this document, specifies the encoding of the public keys of group members, encoded as a CBOR integer. Possible values are the same ones admitted for the 'cs_key_enc' parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

2.3.1.2. Status Properties

The CBOR map MUST include the following status parameters:

- 'active', encoding the CBOR simple value True if the group is currently active, or the CBOR simple value False otherwise. This parameter is defined in Section 4.1 of this specification.

- 'group_name', with value the group name of the OSCORE group encoded as a CBOR text string. This parameter is defined in Section 4.1 of this specification.

- 'group_title', with value either a human-readable description of the group encoded as a CBOR text string, or the CBOR simple value Null if no description is specified. This parameter is defined in Section 4.1 of this specification.

- 'ace-groupcomm-profile', defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], with value "coap_group_oscore_app".

- 'exp', defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm].

- 'joining_uri', with value the URI of the group-membership resource for joining the newly created OSCORE group, encoded as a CBOR text string. This parameter is defined in Section 4.1 of this specification.

The CBOR map MAY include the following status parameters:

- 'group_policies', defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm], and consistent with the format and content defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

- 'as_uri', defined in Section 4.1 of this document, specifies the URI of the Authorization Server associated to the Group Manager for the OSCORE group, encoded as a CBOR text string. Candidate group members will have to obtain an Access Token from that Authorization Server, before starting the joining process with the Group Manager to join the OSCORE group (see [I-D.ietf-ace-key-groupcomm-oscore]).

2.3.2. Default Values

This section defines the default values that the Group Manager assumes for configuration and status parameters.

2.3.2.1. Configuration Parameters

For each configuration parameter, the Group Manager MUST assume a pre-configured default value, if none is specified by the Administrator.

In particular, the Group Manager SHOULD use the same default values defined in Section 18 of [I-D.ietf-ace-key-groupcomm-oscore].

2.3.2.2. Status Parameters

For the following status parameters, the Group Manager MUST assume a pre-configured default value, if none is specified by the Administrator.

o For ’active’, the CBOR simple value False.

o For ’group_title’, the CBOR simple value Null.

2.4. Discovery

The Administrator can discover the group-collection resource from a resource directory, for instance [I-D.ietf-core-resource-directory] and [I-D.hartke-t2trg-coral-reef], or from .well-known/core , by using the resource type "ace.oscore.gm" defined in Section 4.2 of this specification.

The Administrator can discover group-configuration resources for the group-collection resource as specified below in Section 2.6.1 and Section 2.6.2.

2.5. Collection Representation

A list of group configurations is represented as a document containing the corresponding group-configuration resources in the list. Each group-configuration is represented as a link, where the link target is the URI of the group-configuration resource.

The list can be represented as a Link Format document [RFC6690] or a CoRAL document [I-D.ietf-core-coral]. In the latter case, the CoRAL document contains the group-configuration resources in the list as top-level elements. In particular, the link to each group-configuration resource has http://coreapps.org/ace.oscore.gm#item as relation type.

2.6. Interactions

This section describes the operations available on the group-collection resource and the group-configuration resources.

When custom CBOR is used, the Content-Format in messages containing a payload is set to application/ace-groupcomm+cbor, defined in Section 8.2 of [I-D.ietf-ace-key-groupcomm]. Furthermore, the entry labels defined in Section 4.1 MUST be used, when specifying the corresponding configuration and status parameters.

2.6.1. Get All Groups Configurations

The Administrator can send a GET request to the group-collection resource, in order to retrieve the list of the existing OSCORE groups at the Group Manager. This is returned as a list of links to the corresponding group-configuration resources.

Example in Link Format:

```
=> 0.01 GET
   Uri-Path: manage

<= 2.05 Content
   Content-Format: 40 (application/link-format)

   <coap://[2001:db8::ab]/manage/gp1>,
   <coap://[2001:db8::ab]/manage/gp2>,
   <coap://[2001:db8::ab]/manage/gp3>
```

Example in CoRAL:

```
=> 0.01 GET
   Uri-Path: manage

<= 2.05 Content
   Content-Format: TBD1 (application/coral+cbor)

   #using <http://coreapps.org/ace.oscore.gm#>
   #base </manage/>
   item <gp1>
   item <gp2>
   item <gp3>
```

2.6.2. Fetch Group Configurations By Filters

The Administrator can send a FETCH request to the group-collection resource, in order to retrieve the list of the existing OSCORE groups at the Group Manager that fully match a set of specified filter criteria. This is returned as a list of links to the corresponding group-configuration resources.

The set of filter criteria is specified in the request payload as a CBOR map, where possible entry labels are all the ones used for configuration properties (see Section 2.3.1.1), as well as "group_name" and "active" for the corresponding status property (see Section 2.3.1.2).

Entry values are the ones admitted for the corresponding labels in the POST request for creating a group configuration (see Section 2.6.3). A valid request MUST NOT include the same entry multiple times.

Example in custom CBOR and Link Format:

```
=> 0.05 FETCH
   Uri-Path: manage
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
       "alg" : 10,
       "hkdf" : 5
   }

<= 2.05 Content
   Content-Format: 40 (application/link-format)

   <coap://[2001:db8::ab]/manage/gp1>,
   <coap://[2001:db8::ab]/manage/gp2>,
   <coap://[2001:db8::ab]/manage/gp3>
```

Example in CoRAL:

```
   => 0.05 FETCH
      Uri-Path: manage
      Content-Format: TBD1 (application/coral+cbor)

      alg 10
      hkdf 5

   <= 2.05 Content
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      #base </manage/>
      item <gp1>
      item <gp2>
      item <gp3>
```

2.6.3. Create a New Group Configuration

The Administrator can send a POST request to the group-collection resource, in order to create a new OSCORE group at the Group Manager. The request MAY specify the intended group name NAME and group title, and MAY specify pieces of information concerning the group configuration.

The request payload is a CBOR map, whose possible entries are specified in Section 2.3.1. In particular:

- o The CBOR map MAY include any of the configuration parameter defined in Section 2.3.1.1.
- o The CBOR map MAY include any of the status parameter 'group_name', 'group_title', 'exp', 'group_policies', 'as_uri' and 'active' defined in Section 2.3.1.2.
- o The CBOR map MUST NOT include any of the status parameter 'ace-groupcomm-profile' and 'joining_uri' defined in Section 2.3.1.2.

If any of the following occurs, the Group Manager MUST respond with a 4.00 (Bad Request) response, which MAY include additional information to clarify what went wrong.

- o Any of the received parameters is specified multiple times.
- o Any of the received parameters is not recognized, or not valid, or not consistent with respect to other related parameters.
- o The 'group_name' parameter specifies the group name of an already existing OSCORE group.

- The Group Manager does not trust the Authorization Server with URI specified in the 'as_uri' parameter, and has no alternative Authorization Server to consider for the OSCORE group to create.

After a successful processing of the request above, the Group Manager performs the following actions.

First, the Group Manager creates a new group-configuration resource, accessible to the administrator at /manage/NAME , where NAME is the group name as either indicated in the parameter 'group_name' of the request or uniquely assigned by the Group Manager. The values specified in the request are used as group configuration information for the newly created OSCORE group. For each configuration parameter not specified in the request, the Group Manager MUST assume the default value specified in Section 2.3.2.

After that, the Group Manager creates a new group-membership resource, accessible to joining nodes and future group members at group-oscore/NAME , as specified in [I-D.ietf-ace-key-groupcomm-oscore]. In particular, the Group Manager will rely on the current group configuration to build the Joining Response message defined in Section 5.4 of [I-D.ietf-ace-key-groupcomm-oscore], when handling the joining of a new group member. Furthermore, the Group Manager generates the following pieces of information, and assigns them to the newly created OSCORE group:

- The OSCORE Master Secret.

- The OSCORE Master Salt (optionally).

- The OSCORE ID Context, acting as Group ID, which MUST be unique within the set of OSCORE groups under the Group Manager.

Finally, the Group Manager replies to the Administrator with a 2.01 (Created) response. The Location-Path option MUST be included in the response, indicating the location of the just created group-configuration resource. The response MUST NOT include a Location-Query option. The response payload is a CBOR map, which MUST include the following parameters:

- 'group_name', with value the group name of the OSCORE group encoded as a CBOR text string. This value can be different from the group name possibly specified by the Administrator in the POST request, and reflects the final choice of the Group Manager as 'group_name' status property for the OSCORE group.

- 'joining_uri', with value the URI of the group-membership resource for joining the newly created OSCORE group, encoded as a CBOR text string.

- 'as_uri', with value the URI of the Authorization Server associated to the Group Manager for the newly created OSCORE group, encoded as a CBOR text string. This value can be different from the URI possibly specified by the Administrator in the POST request, and reflects the final choice of the Group Manager as 'as_uri' status property for the OSCORE group.

At this point, the Group Manager can register the link to the group-membership resource with URI specified in 'joining_uri' to the CoRE Resource Directory [I-D.ietf-core-resource-directory], as defined in Section 2 of [I-D.tiloca-core-oscore-discovery].

Alternatively, the Administrator can perform the registration to the Resource Directory on behalf of the Group Manager, acting as Commissioning Tool. The Administrator considers the following when specifying additional information for the link to register.

- The name of the OSCORE group MUST take the value specified in 'group_name' from the 2.01 (Created) response above.

- If present, parameters describing the cryptographic algorithms used in the group MUST follow the values that the Administrator specified in the POST request above, or the corresponding default values specified in Section 2.3.2.1 otherwise.

- If also registering a related link to the Authorization Server associated to the OSCORE group, the related link MUST have the URI specified in 'as_uri' from the 2.01 (Created) response above.

Example in custom CBOR:

```
   => 0.02 POST
      Uri-Path: manage
      Content-Format: TBD2 (application/ace-groupcomm+cbor)

      {
        "alg" : 10,
        "hkdf" : 5,
        "active" : True,
        "group_title" : "rooms 1 and 2",
        "as_uri" : "coap://as.example.com/token"
      }

   <= 2.01 Created
      Location-Path: manage
      Location-Path: gp4
      Content-Format: TBD2 (application/ace-groupcomm+cbor)

      {
        "group_name" : "gp4",
        "joining_uri" : "coap://[2001:db8::ab]/group-oscore/gp4/",
        "as_uri" : "coap://as.example.com/token"
      }
```

Example in CoRAL:

```
   => 0.02 POST
      Uri-Path: manage
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      alg 10
      hkdf 5
      active True
      group_title "rooms 1 and 2"
      as_uri <coap://as.example.com/token>

   <= 2.01 Created
      Location-Path: manage
      Location-Path: gp4
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      group_name "gp4"
      joining_uri <coap://[2001:db8::ab]/group-oscore/gp4/>
      as_uri <coap://as.example.com/token>
```

2.6.4. Retrieve a Group Configuration

The Administrator can send a GET request to the group-configuration resource manage/NAME associated to an OSCORE group with group name NAME, in order to retrieve the current configuration of that group.

After a successful processing of the request above, the Group Manager replies to the Administrator with a 2.05 (Content) response. The response has as payload the representation of the group configuration as specified in Section 2.3.1. The exact content of the payload reflects the current configuration of the OSCORE group. This includes both configuration properties and status properties.

Example in custom CBOR:

```
=> 0.01 GET
   Uri-Path: manage
   Uri-Path: gp4

<= 2.05 Content
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
     "alg" : 10,
     "hkdf" : 5,
     "cs_alg" : -8,
     "cs_params" : [[1], [1, 6]],
     "cs_key_params" : [1, 6],
     "cs_key_enc" : 1,
     "active" : True,
     "group_name" : "gp4",
     "group_title" : "rooms 1 and 2",
     "ace-groupcomm-profile" : "coap_group_oscore_app",
     "exp" : "1360289224",
     "joining_uri" : "coap://[2001:db8::ab]/group-oscore/gp4/",
     "as_uri" : "coap://as.example.com/token"
   }
```

Example in CoRAL:

```
   => 0.01 GET
      Uri-Path: manage
      Uri-Path: gp4

   <= 2.05 Content
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      alg 10
      hkdf 5
      cs_alg -8
      cs_params.alg_capab.key_type 1
      cs_params.key_type_capab.key_type 1
      cs_params.key_type_capab.curve 6
      cs_key_params.key_type 1
      cs_key_params.curve 6
      cs_key_enc 1
      active True
      group_name "gp4"
      group_title "rooms 1 and 2"
      ace-groupcomm-profile "coap_group_oscore_app"
      exp "1360289224"
      joining_uri <coap://[2001:db8::ab]/group-oscore/gp4/>
      as_uri <coap://as.example.com/token>
```

2.6.5. Update a Group Configuration

The Administrator can send a PUT request to the group-configuration resource associated to an OSCORE group, in order to update the current configuration of that group. The payload of the request has the same format of the POST request defined in Section 2.6.3, with the exception of the status parameter 'group_name' that MUST NOT be included.

The error handling for the PUT request is the same as for the POST request defined in Section 2.6.3. If no error occurs, the Group Manager performs the following actions.

First, the Group Manager updates the configuration of the OSCORE group, consistently with the values indicated in the PUT request from the Administrator. For each configuration parameter not specified in the PUT request, the Group Manager MUST use the default value specified in Section 2.3.2. From then on, the Group Manager relies on the latest update configuration to build the Joining Response message defined in Section 5.4 of [I-D.ietf-ace-key-groupcomm-oscore], when handling the joining of a new group member.

Then, the Group Manager replies to the Administrator with a 2.04 (Changed) response. The payload of the response has the same format of the 2.01 (Created) response defined in Section 2.6.3.

If the link to the group-membership resource was registered in the Resource Directory (see [I-D.ietf-core-resource-directory]), the GM is responsible to refresh the registration, as defined in Section 3 of [I-D.tiloca-core-oscore-discovery].

Alternatively, the Administrator can update the registration to the Resource Directory on behalf of the Group Manager, acting as Commissioning Tool. The Administrator considers the following when specifying additional information for the link to update.

- The name of the OSCORE group MUST take the value specified in 'group_name' from the 2.04 (Changed) response above.

- If present, parameters describing the cryptographic algorithms used in the group MUST follow the values that the Administrator specified in the POST request above, or the corresponding default values specified in Section 2.3.2.1 otherwise.

- If also registering a related link to the Authorization Server associated to the OSCORE group, the related link MUST have the URI specified in 'as_uri' from the 2.04 (Changed) response above.

Example in custom CBOR:

```
=> PUT
   Uri-Path: manage
   Uri-Path: gp4
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
     "alg" : 11 ,
     "hkdf" : 5
   }

<= 2.04 Changed
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
     "group_name" : "gp4",
     "joining_uri" : "coap://[2001:db8::ab]/group-oscore/gp4/",
     "as_uri" : "coap://as.example.com/token"
   }
```

Example in CoRAL:

```
   => PUT
      Uri-Path: manage
      Uri-Path: gp4
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      alg 11
      hkdf 5

   <= 2.04 Changed
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/ace.oscore.gm#>
      group_name "gp4"
      joining_uri <coap://[2001:db8::ab]/group-oscore/gp4/>
      as_uri <coap://as.example.com/token>
```

2.6.5.1. Effects on Joining Nodes

If the value of the status parameter 'active' is changed from True to False, the Group Manager MUST stop admitting new members in the group. In particular, upon receiving a Joining Request (see Section 5.3 of [I-D.ietf-ace-key-groupcomm-oscore]), the Group Manager MUST respond with a 5.03 (Service Unavailable) response to the joining node, and MAY include additional information to clarify what went wrong.

If the value of the status parameter 'active' is changed from False to True, the Group Manager resumes admitting new members in the group, by processing their Joining Requests (see Section 5.3 of [I-D.ietf-ace-key-groupcomm-oscore]).

2.6.5.2. Effects on the Group Members

After having updated a group configuration, the Group Manager informs the group members, over the pairwise secure communication channels established when joining the OSCORE group (see Section 5 of [I-D.ietf-ace-key-groupcomm-oscore]).

To this end, the Group Manager can individually target the 'control_path' URI path of each group member (see Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm]), if provided by the intended recipient upon joining the group (see Section 5 of [I-D.ietf-ace-key-groupcomm-oscore]). Alternatively, group members can subscribe for updates to the group-membership resource of the OSCORE group, e.g. by using CoAP Observe [RFC7641].

Every group member, upon learning that the group has been deactivated (i.e. 'active' has value False), SHOULD stop communicating in the OSCORE group.

Every group member, upon learning that the group has been reactivated (i.e. 'active' has value True again), can resume communicating in the OSCORE group.

Every group member, upon receiving updated values for 'alg' and 'hkdf', MUST either:

- o Leave the group (see Section 14 of [I-D.ietf-ace-key-groupcomm-oscore]), e.g. if not supporting the indicated new algorithms; or

- o Use the new parameter values, and accordingly re-derive the OSCORE Security Context for the group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

Every group member, upon receiving updated values for 'cs_alg', 'cs_params', 'cs_key_params' and 'cs_key_enc', MUST either:

- o Leave the group, e.g. if not supporting the indicated new algorithm, parameters and encoding; or

- o Leave the group and rejoin it (see Section 5 of [I-D.ietf-ace-key-groupcomm-oscore]), providing the Group Manager with a public key which is compatible with the indicated new algorithm, parameters and encoding; or

- o Use the new parameter values, and, if required, provide the Group Manager with a new public key to use in the group, as compatible with the indicated parameters (see Section 10 of [I-D.ietf-ace-key-groupcomm-oscore]).

2.6.6. Delete a Group Configuration

The Administrator can send a DELETE request to the group-configuration resource, in order to delete that group. A group deletion would be successful only on an inactive group.

That is, the DELETE request actually yields a successful deletion of the group, only if the corresponding status parameter 'active' has current value False. The administrator can ensure that, by first performing an update of the group-configuration resource associated to the group (see Section 2.6.5), and setting the corresponding status parameter 'active' to False.

If, upon receiving the DELETE request, the current value of the status parameter 'active' is True, the Group Manager MUST respond with a 4.09 (Conflict) response, which MAY include additional information to clarify what went wrong.

After a successful processing of the request above, the Group Manager performs the following actions.

First, the Group Manager deletes the OSCORE group and deallocates both the group-configuration resource as well as the group-membership resource.

Then, the Group Manager replies to the Administrator with a 2.02 (Deleted) response.

Example:

```
=> DELETE
   Uri-Path: manage
   Uri-Path: gp4

<= 2.02 Deleted
```

2.6.6.1. Effects on the Group Members

After having deleted a group, the Group Manager can inform the group members by means of the following two methods. When contacting a group member, the Group Manager uses the pairwise secure communication channel established with that member during its joining process (see Section 5 of [I-D.ietf-ace-key-groupcomm-oscore]).

- The Group Manager sends an individual request message to each group member, targeting the respective resource used to perform the group rekeying process (see Section 16 of [I-D.ietf-ace-key-groupcomm-oscore]). The Group Manager uses the same format of the Joining Response message in Section 5.4 of [I-D.ietf-ace-key-groupcomm-oscore], where only the parameters 'gkty', 'key', and 'ace-groupcomm-profile' are present, and the 'key' parameter is empty.

- A group member may subscribe for updates to the group-membership resource of the group. In particular, if this relies on CoAP Observe [RFC7641], a group member would receive a 4.04 (Not Found) notification response from the Group Manager, since the group-configuration resource has been deallocated upon deleting the group.

When being informed about the group deletion, a group member deletes the OSCORE Security Context that it stores as associated to that group, and possibly deallocates any dedicated control resource intended for the Group Manager that it has for that group.

3. Security Considerations

Security considerations are inherited from the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and from the specific transport ace-groupcomm-profile of ACE used between the Administrator and the Group Manager, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

4. IANA Considerations

This document has the following actions for IANA.

4.1. ACE Groupcomm Parameters Registry

IANA is asked to register the following entries in the "ACE Groupcomm Parameters" Registry defined in Section 8.5 of [I-D.ietf-ace-key-groupcomm].

Name	CBOR Key	CBOR Type	Reference
hkdf	TBD3	tstr	[[this document]]
alg	TBD4	tstr	[[this document]]
cs_alg	TBD5	tstr / int	[[this document]]
cs_params	TBD6	array	[[this document]]
cs_key_params	TBD7	array	[[this document]]
cs_key_enc	TBD8	int	[[this document]]
active	TBD9	simple type	[[this document]]
group_name	TBD10	tstr	[[this document]]
group_title	TBD11	tstr / simple type	[[this document]]
joining_uri	TBD12	tstr	[[this document]]
as_uri	TBD13	tstr	[[this document]]

4.2. Resource Types

IANA is asked to enter the following value into the Resource Type (rt=) Link Target Attribute Values subregistry within the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690].

Value	Description	Reference
ace.oscore.gm	Group-collection resource of an OSCORE Group Manager	[[this document]]

5. References

5.1. Normative References

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

[COSE.Elliptic.Curves]
IANA, "COSE Elliptic Curves", <https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>.

[COSE.Key.Types]
IANA, "COSE Key Types", <https://www.iana.org/assignments/cose/cose.xhtml#key-type>.

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-08 (work in progress), July 2020.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-08 (work in progress), July 2020.

[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-35 (work in progress), June 2020.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-11 (work in progress), June 2020.

[I-D.ietf-core-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 2020.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-09 (work in progress), June 2020.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", draft-ietf-cose-rfc8152bis-algs-11 (work in progress), July 2020.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", draft-ietf-cose-rfc8152bis-struct-11 (work in progress), July 2020.

[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>.

[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>.

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

[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>.

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

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

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

[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>.

[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>.

5.2. Informative References

[I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-03 (work in progress), March 2020.

[I-D.hartke-t2trg-coral-reef]
Hartke, K., "Resource Discovery in Constrained RESTful Environments (CoRE) using the Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-reef-04 (work in progress), May 2020.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-12 (work in progress), July 2020.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-24 (work in progress), March 2020.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-05 (work in progress), March 2020.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

Acknowledgments

The authors sincerely thank Christian Amsuess, Carsten Bormann and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Peter van der Stok
Consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: klaus.hartke@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

M. Tiloca
RISE AB
L. Seitz
Combitech
F. Palombini
Ericsson AB
S. Echeverria
G. Lewis
CMU SEI

November 02, 2020

Notification of Revoked Access Tokens in the Authentication and Authorization for Constrained Environments (ACE) Framework
draft-tiloca-ace-revoked-token-notification-03

Abstract

This document specifies a method of the Authentication and Authorization for Constrained Environments (ACE) framework, which allows an Authorization Server to notify Clients and Resource Servers (i.e., registered devices) about revoked Access Tokens. The method relies on resource observation for the Constrained Application Protocol (CoAP), with Clients and Resource Servers observing a Token Revocation List on the Authorization Server. Resulting unsolicited notifications of revoked Access Tokens complement alternative approaches such as token introspection, while not requiring additional endpoints on Clients and Resource Servers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction   3
  1.1. Terminology   3
2. Protocol Overview   5
3. Token Hash   7
4. The TRL Resource   8
  4.1. Update of the TRL Resource   9
5. The TRL Endpoint   9
  5.1. Full Query of the TRL 10
  5.2. Diff Query of the TRL 10
6. Upon Registration 12
7. Notification of Revoked Tokens 13
8. Interaction Examples 13
  8.1. Full Query with Observation 14
  8.2. Diff Query with Observation 15
  8.3. Full Query with Observation and Additional Diff Query . . 17
9. Security Considerations 20
10. IANA Considerations 21
11. References 21
  11.1. Normative References 21
  11.2. Informative References 22
Appendix A. Usage of the Series Transfer Pattern 22
Appendix B. Usage of the "Cursor" Pattern 23
  B.1. Full Query Request 24
  B.2. Full Query Response 24
  B.3. Diff Query Request 24
  B.4. Diff Query Response 25
    B.4.1. Empty Collection 25
    B.4.2. Cursor Not Specified in the Diff Query Request . . . 25
    B.4.3. Cursor Specified in the Diff Query Request 26
Acknowledgments 27
Authors' Addresses 28

1. Introduction

Authentication and Authorization for Constrained Environments (ACE) [I-D.ietf-ace-oauth-authz] is a framework that enforces access control on IoT devices acting as Resource Servers. In order to use ACE, both Clients and Resource Servers have to register with an Authorization Server and become a registered device. Once registered, a Client can send a request to the Authorization Server, to obtain an Access Token for a Resource Server. For a Client to access the Resource Server, the Client must present the issued Access Token at the Resource Server, which then validates and stores it.

Even though Access Tokens have expiration times, there are circumstances by which an Access Token may need to be revoked before its expiration time, such as: (1) a registered device has been compromised, or is suspected of being compromised; (2) a registered device is decommissioned; (3) there has been a change in the ACE profile for a registered device; (4) there has been a change in access policies for a registered device; and (5) there has been a change in the outcome of policy evaluation for a registered device (e.g., if policy assessment depends on dynamic conditions in the execution environment, the user context, or the resource utilization).

As discussed in Section 6.1 of [I-D.ietf-ace-oauth-authz], only client-initiated revocation is currently specified [RFC7009] for OAuth 2.0 [RFC6749], based on the assumption that Access Tokens in OAuth are issued with a relatively short lifetime. However, this may not be the case for constrained, intermittently connected devices, that need Access Tokens with relatively long lifetimes.

This document specifies a method for allowing registered devices to access and observe a Token Revocation List (TRL) resource on the Authorization Server, in order to get an updated list of revoked, but yet not expired, pertaining Access Tokens. In particular, registered devices rely on resource observation [RFC7641] for the Constrained Application Protocol (CoAP) [RFC7252]. The benefits of this method are that it complements token introspection and does not require any additional endpoints on the registered devices.

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 the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], as well as with terms and concepts related to CBOR Web Tokens (CWTs) [RFC8392], and JSON Web Tokens (JWTs) [RFC7519]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client, Resource Server, and Authorization Server.

Readers are also expected to be familiar with the terms and concepts related to CBOR [I-D.ietf-cbor-7049bis], JSON [RFC8259], the CoAP protocol [RFC7252], CoAP Observe [RFC7641], and the use of hash functions to name objects as defined in [RFC6920].

Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the Authorization Server, and /authz-info at the Resource Server. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol."

This specification also refers to the following terminology.

- Token hash: identifier of an Access Token, in binary format encoding. The token hash has no relation to other possibly used token identifiers, such as the "cti" (CWT ID) claim of CBOR Web Tokens (CWTs) [RFC8392].

- Token Revocation List (TRL): a collection of token hashes, in which the corresponding Access Tokens have been revoked but are not expired yet.

- TRL resource: a resource on the Authorization Server, with a TRL as its representation.

- TRL endpoint: an endpoint at the Authorization Server associated to the TRL resource. The default name of the TRL endpoint in a url-path is '/revoke/trl'. Implementations are not required to use this name, and can define their own instead.

- Registered device: a device registered at the Authorization Server, i.e. as a Client, or a Resource Server, or both. A registered device acts as caller of the TRL endpoint.

- Administrator: entity authorized to get full access to the TRL at the Authorization Server, and acting as caller of the TRL endpoint. An administrator is not necessarily a registered device as defined above, i.e. a Client requesting Access Tokens or a Resource Server consuming Access Tokens. How the administrator authorization is established and verified is out of the scope of this specification.

- Pertaining Access Token:

  * With reference to an administrator, an Access Token issued by the Authorization Server.

  * With reference to a registered device, an Access Token intended to be owned by that device. An Access Token pertains to a Client if the Authorization Server has issued the Access Token and provided it to that Client. An Access Token pertains to a Resource Server if the Authorization Server has issued the Access Token to be consumed by that Resource Server.

2. Protocol Overview

This protocol defines how a CoAP-based Authorization Server informs Clients and Resource Servers, i.e. registered devices, about revoked Access Tokens. How the relationship between the registered device and the Authorization Server is established is out of the scope of this specification.

At a high level, the steps of this protocol are as follows.

- Upon startup, the Authorization Server creates a TRL resource. At any point in time, the TRL resource represents the list of all revoked Access Tokens issued by the Authorization Server that are yet not expired.

- When a device registers at the Authorization Server, it receives the url-path to the TRL resource. After the registration procedure is finished, the registered device sends an Observation Request to that TRL resource as described in [RFC7641], i.e. a GET request with an Observe option set to 0 (register). Upon receiving the request, the Authorization Server adds the registered device to the list of observers of the TRL resource. At any time, the registered device can send a GET request to the TRL endpoint, in order to get the current list of pertaining revoked Access Tokens.

- When an Access Token is revoked, the Authorization Server adds the corresponding token hash to the TRL. Also, when a revoked Access Token eventually expires, the Authorization Server removes the corresponding token hash from the TRL. In either case, after updating the TRL, the Authorization Server sends Observe Notifications as described in [RFC7641]. That is, one Observe Notification is sent to each registered device the Access Token pertains to, and specifies the current updated list of token hashes in the portion of the TRL pertaining to that device.

- o An administrator can observe and access the TRL like a registered device, while getting the full updated representation of the TRL.

Figure 1 provides a high-level overview of the service provided by this protocol. Each dotted line associated to a pair of registered devices indicates the Access Token that they both own. In particular, Figure 1 shows the Observe Notifications sent by the Authorization Server to four registered devices and one administrator, upon revocation of the issued Access Tokens t1, t2 and t3, with token hash th1, th2 and th3, respectively.

```
                           +---------------+
                           |               |
                           | Authorization |
                           |    Server     |
                           |               |
                           +-------o-------+
                       revoke/trl  |  TRL: {th1,th2,th3}
                                   |
                                   |
    +-----------------+------------+------------+------------+
    |                 |            |            |            |
    |                 |            |            |            |
    | th1,th2,th3     | th1,th2    | th1        | th3        | th2,th3
    v                 v            v            v            v
   +---------------+ +----------+ +----------+ +----------+ +----------+
   |               | |          | |          | |          | |          |
   | Administrator | | Client 1 | | Resource | | Client 2 | | Resource |
   |               | |          | | Server 1 | |          | | Server 2 |
   +---------------+ +----------+ +----------+ +----------+ +----------+
                        :    :        :           :            :    :
                        :    :   t1   :           :     t3     :    :
                        :    :........:           :............:    :
                        :                                           :
                        :                    t2                     :
                        :...........................................:
```

Figure 1: Protocol Overview

More detailed examples describing the protocol flow and message exchange between the Authorization Server and a registered device are provided in Section 8.

3. Token Hash

The token hash of an Access Token is computed as follows.

1. The Authorization Server defines ENCODED_TOKEN, as the content of the 'access_token' parameter in the Authorization Server response (see Section 5.6.2 of [I-D.ietf-ace-oauth-authz]), where the Access Token was included and returned to the requesting Client.

   Note that the content of the 'access_token' parameter is either:

   * A CBOR byte string, if the Access Token was transported using CBOR. With reference to the example in Figure 2, and assuming the string's length in bytes to be 119 (i.e., 0x77 in hexadecimal), then ENCODED_TOKEN takes the bytes {0x58 0x77 0xd0 0x83 0x44 0xa1 ...}, i.e. the raw content of the parameter 'access_token'.

   * A text string, if the Access Token was transported using JSON. With reference to the example in Figure 3, ENCODED_TOKEN takes "2YotnFZFEjr1zCsicMWpAA", i.e. the raw content of the parameter 'access_token'.

2. The Authorization Server defines HASH_INPUT as follows.

   * If CBOR was used to transport the Access Token (as a CWT or JWT), HASH_INPUT takes the same value of ENCODED_TOKEN.

   * If JSON was used to transport the Access Token (as a CWT or JWT), HASH_INPUT takes the serialization of ENCODED_TOKEN.

     In either case, HASH_INPUT results in the binary representation of the content of the 'access_token' parameter from the Authorization Server response.

3. The Authorization Server generates a hash value of HASH_INPUT as per Section 6 of [RFC6920]. The resulting output in binary format is used as the token hash. Note that the used binary format embeds the identifier of the used hash function, in the first byte of the computed token hash.

   The specifically used hash function MUST be collision-resistant on byte-strings, and MUST be selected from the "Named Information Hash Algorithm" Registry [Named.Information.Hash.Algorithm].

   The Authorization Server specifies the used hash function to registered devices during their registration procedure (see Section 6).

```
   2.01 Created
   Content-Format: application/ace+cbor
   Max-Age: 85800
   Payload:
   {
      access_token : h'd08344a1...'
      (remainder of the Access Token omitted for brevity)
      token_type : pop,
      expires_in : 86400,
      profile    : coap_dtls,
      (remainder of the response omitted for brevity)
   }
```

Figure 2: Example of Authorization Server response using CBOR

```
   HTTP/1.1 200 OK
   Content-Type: application/json
   Cache-Control: no-store
   Pragma: no-cache
   Payload:
   {
      "access_token" : "2YotnFZFEjr1zCsicMWpAA"
      (remainder of the Access Token omitted for brevity)
      "token_type" : "pop",
      "expires_in" : 86400,
      "profile"    : "coap_dtls",
      (remainder of the response omitted for brevity)
   }
```

Figure 3: Example of Authorization Server response using JSON

4. The TRL Resource

Upon startup, the Authorization Server creates a single TRL resource, encoded as a CBOR array.

Each element of the array is a token hash, encoded as a CBOR byte string. The order of the token hashes in the CBOR array is irrelevant, and the CBOR array MUST be treated as a set in which the order has no significant meaning.

The TRL is initialized as empty, i.e. the initial content of the TRL resource representation MUST be an empty CBOR array.

4.1. Update of the TRL Resource

The Authorization Server updates the TRL in the following two cases.

- o When a non-expired Access Token is revoked, the token hash of the Access Token is added to the TRL resource representation. That is, a CBOR byte string encoding the token hash is added to the CBOR array used as TRL resource representation.
- o When a revoked Access Token expires, the token hash of the Access Token is removed from the TRL resource representation. That is, the CBOR byte string encoding the token hash is removed from the CBOR array used as TRL resource representation.

5. The TRL Endpoint

Consistent with Section 6.5 of [I-D.ietf-ace-oauth-authz], all communications between a caller of the TRL endpoint and the Authorization Server MUST be encrypted, as well as integrity and replay protected. Furthermore, responses from the Authorization Server to the caller MUST be bound to the caller's request.

The Authorization Server MUST implement measures to prevent access to the TRL endpoint by entities other than registered devices and authorized administrators.

The TRL endpoint supports only the GET method, and provides two types of query of the TRL.

- o Full query: the Authorization Server returns the token hashes of the revoked Access Tokens currently in the TRL and pertaining to the requester. The processing of a full query and the related response format are defined in Section 5.1.
- o Diff query: the Authorization Server returns a list of diff entries. Each entry is related to one of the most recent updates, in the portion of the TRL pertaining to the requester. In particular, the entry associated to one of such updates contains a list of token hashes, such that: i) the corresponding revoked Access Tokens pertain to the requester; and ii) they were added to or removed from the TRL at that update. The processing of a diff query and the related response format are defined in Section 5.2.

The TRL endpoint allows the following query parameter in a GET request.

- o 'diff': if included, it indicates to perform a diff query of the TRL. Its value MUST be either: i) 0, indicating that a (notification) response should include as many diff entries as the Authorization Server can provide; or ii) a positive integer greater than 0, indicating the maximum number of diff entries that a (notification) response should include.

5.1. Full Query of the TRL

In order to produce a (notification) response to a GET request asking for a full query of the TRL, the Authorization Server performs the following actions.

1. From the current TRL resource representation, the Authorization Server builds a set HASHES, such that:

   * If the requester is a registered device, HASHES specifies the token hashes of the Access Tokens pertaining to that registered device. The Authorization Server can use the authenticated identity of the registered device to perform the necessary filtering on the TRL resource representation.

   * If the requester is an administrator, HASHES specifies all the token hashes in the current TRL resource representation.

2. The Authorization Server sends a 2.05 (Content) Response to the requester, with a CBOR array as payload. Each element of the array specifies one of the token hashes from the set HASHES, encoded as a CBOR byte string.

   The order of the token hashes in the CBOR array is irrelevant, i.e. the CBOR array MUST be treated as a set in which the order has no significant meaning.

5.2. Diff Query of the TRL

In order to produce a (notification) response to a GET request asking for a diff query of the TRL, the Authorization Server performs the following actions.

1. The Authorization Server defines the positive integer NUM. If the value N specified in the query parameter 'diff' of the GET request is equal to 0 or greater than a pre-defined positive integer N_MAX, then NUM takes the value of N_MAX. Otherwise, NUM takes N.

2. The Authorization Server prepares U = min(NUM, SIZE) diff entries, where SIZE <= N_MAX is the number of TRL updates pertaining to the requester and currently stored at the Authorization Server. That is, the diff entries are related to

the U most recent TRL updates pertaining to the requester. In particular, the first entry refers to the most recent of such updates, the second entry refers to the second from last of such updates, and so on.

Each diff entry is a CBOR array 'diff-entry', which includes the following two elements.

* The first element is a CBOR array 'removed'. Each element of the array is a CBOR byte string encoding the token hash of an Access Token, that pertained to the requester and that was removed from the TRL during the update associated to the diff entry.

* The second element is a CBOR array 'added'. Each element of the array is a CBOR byte string encoding the token hash of an Access Token, that pertains to the requester and that was added to the TRL during the update associated to the diff entry.

The order of the token hashes in the CBOR arrays 'removed' and 'added' is irrelevant. That is, the CBOR arrays 'removed' and 'added' MUST be treated as a set in which the order of elements has no significant meaning.

3. The Authorization Server prepares a 2.05 (Content) Response for the requester, with a CBOR array 'diff' of U elements as payload. Each element of the array specifies one of the CBOR arrays 'diff-entry' prepared at point 2 as diff entries.

   Within the CBOR array 'diff', the CBOR arrays 'diff-entry' MUST be sorted to reflect the corresponding updates to the TRL in reverse chronological order. That is, the first 'diff-entry' element of 'diff' relates to the most recent update to the portion of the TRL pertaining to the requester.

The CDDL definition [RFC8610] of the CBOR array 'diff' formatted as in the response from the Authorization Server is provided below.

```
token-hash = bytes
trl-patch = [* token-hash]
diff-entry = [removed: trl-patch, added: trl-patch]
diff = [* diff-entry]
```

Figure 4: CDDL definition of the response payload following a Diff Query request to the TRL endpoint

If the Authorization Server supports diff queries:

- o The Authorization Server MUST keep track of N_MAX most recent updates to the portion of the TRL that pertains to each caller of the TRL endpoint. The particular method to achieve this is implementation-specific.

- o When SIZE is equal to N_MAX, and a new TRL update occurs as pertaining to a registered device, the Authorization Server MUST first delete the oldest stored update for that device, before storing this latest update as the most recent one for that device.

- o The Authorization Server SHOULD provide registered devices and administrators with the value of N_MAX, upon their registration (see Section 6).

If the Authorization Server does not support diff queries, it proceeds as when processing a full query (see Section 5.1).

Appendix A discusses how the diff query of the TRL is in fact a usage example of the Series Transfer Pattern defined in [I-D.bormann-t2trg-stp].

Appendix B discusses how the diff query of the TRL can be further improved by using the "Cursor" pattern defined in Section 3.3 of [I-D.bormann-t2trg-stp].

6. Upon Registration

During the registration process at the Authorization Server, an administrator or a registered device receives the following information as part of the registration response.

- o The url-path to the TRL endpoint at the Authorization Server.

- o The hash function used to compute token hashes. This is specified as an integer or a text string, taking value from the "ID" or "Hash Name String" column of the "Named Information Hash Algorithm" Registry [Named.Information.Hash.Algorithm], respectively.

- o Optionally, a positive integer N_MAX, if the Authorization Server supports diff queries of the TRL resource (see Section 5.2).

After the registration procedure is finished, the administrator or registered device performs a GET request to the TRL resource, including the CoAP Observe option set to 0 (register), in order to start an observation of the TRL resource at the Authorization Server, as per Section 3.1 of [RFC7641]. The GET request can express the wish for a full query (see Section 5.1) or a diff query (see Section 5.2) of the TRL.

The Authorization Server MUST reply using the CoAP response code 2.05 (Content) and including the CoAP Observe option in the response. The payload of the response is formatted as defined in Section 5.1 or in Section 5.2, in case the GET request was for a full query or a diff query of the TRL, respectively.

7. Notification of Revoked Tokens

In the case the TRL is updated (see Section 4.1), the Authorization Server sends Observe Notifications to every observer of the TRL resource. Observe Notifications are sent as per Section 4.2 of [RFC7641].

The payload of each Observe Notification is formatted as defined in Section 5.1 or in Section 5.2, in case the original Observation Request was for a full query or a diff query of the TRL, respectively.

Furthermore, an administrator or a registered device can send additional GET requests to the TRL endpoint at any time, in order to retrieve the token hashes of the pertaining revoked Access Tokens. When doing so, the caller of the TRL endpoint can perform a full query (see Section 5.1) or a diff query (see Section 5.2).

8. Interaction Examples

This section provides examples of interactions between a Resource Server RS as registered device and an Authorization Server AS. The Authorization Server supports both full query and diff query of the TRL, as defined in Section 5.1 and Section 5.2, respectively.

The details of the registration process are omitted, but it is assumed that the Resource Server sends an unspecified payload to the Authorization Server, which replies with a 2.01 (Created) response.

The payload of the registration response is a CBOR map, which includes the following entries:

- o a "trl" parameter, specifying the path of the TRL resource;
- o a "trl-hash" parameter, specifying the hash function used to computed token hashes as defined in Section 3;
- o an "n-max" parameter, specifying the value of N_MAX, i.e. the maximum number of TRL updates pertaining to each registered device

that the Authorization Server retains for that device (see Section 5.2);

o possible further parameters related to the registration process.

Furthermore, 'h(x)' refers to the hash function used to compute the token hashes, as defined in Section 3 of this specification and according to [RFC6920]. Assuming the usage of CWTs transported in CBOR, 'bstr.h(t1)' and 'bstr.h(t2)' denote the byte-string representations of the token hashes for the Access Tokens t1 and t2, respectively.

8.1. Full Query with Observation

Figure 5 shows an example interaction between the Resource Server RS and the Authorization Server AS, considering a CoAP observation and a full query of the TRL.

```
                 RS                                     AS
                 |                                       |
                 | Registration: POST                    |
                 +-------------------------------------->|
                 |                                       |
                 |<--------------------------------------+
                 |           2.01 CREATED                |
                 |            Payload: {                 |
                 |               ...                     |
                 |               "trl" = "revoke/trl",   |
                 |               "trl-hash" = "sha-256", |
                 |               "n-max" = 10            |
                 |            }                          |
                 |                                       |
                 | GET Observe: 0                        |
                 |  coap://example.as.com/revoke/trl/    |
                 +-------------------------------------->|
                 |                                       |
                 |<--------------------------------------+
                 |              2.05 CONTENT Observe: 42 |
                 |               Payload: []             |
                 |                   .                   |
                 |                   .                   |
                 |                   .                   |
                 |                                       |
                 |    (Access Tokens t1 and t2 issued    |
                 |   and successfully submitted to RS)   |
                 |                   .                   |
                 |                   .                   |
                 |                   .                   |
                 |                                       |
                 |     (Access Token t1 is revoked)      |
                 |                                       |
                 |<--------------------------------------+
                 |              2.05 CONTENT Observe: 53 |
                 |               Payload: [bstr.h(t1)]   |
                 |                   .                   |
                 |                   .                   |
                 |                   .                   |
                 |                                       |
                 |     (Access Token t2 is revoked)      |
                 |                                       |
                 |<--------------------------------------+
                 |              2.05 CONTENT Observe: 64 |
                 |               Payload: [bstr.h(t1),   |
                 |                         bstr.h(t2)]   |
                 |                   .                   |
                 |                   .                   |
                 |                   .                   |
                 |                                       |
                 |       (Access Token t1 expires)       |
                 |                                       |
                 |<--------------------------------------+
                 |              2.05 CONTENT Observe: 75 |
                 |               Payload: [bstr.h(t2)]   |
                 |                   .                   |
                 |                   .                   |
                 |                   .                   |
                 |                                       |
                 |       (Access Token t2 expires)       |
                 |                                       |
                 |<--------------------------------------+
                 |              2.05 CONTENT Observe: 86 |
                 |               Payload: []             |
                 |                                       |
```

Figure 5: Interaction for Full Query with Observation

8.2. Diff Query with Observation

Figure 6 shows an example interaction between the Resource Server RS and the Authorization Server AS, considering a CoAP observation and a diff query of the TRL.

The Resource Server indicates N=3 as value of the query parameter "diff", i.e. as the maximum number of diff entries to be specified in a response from the Authorization Server.

```
RS                                              AS
|                                                |
| Registration: POST                             |
+----------------------------------------------->|
|                                                |
|<-----------------------------------------------+
|                  2.01 CREATED                  |
|                   Payload: {                   |
|                      ...                       |
|                      "trl" = "revoke/trl",     |
|                      "trl-hash" = "sha-256",   |
|                      "n-max" = 10              |
|                   }                            |
|                                                |
| GET Observe: 0                                 |
|  coap://example.as.com/revoke/trl?diff=3       |
+----------------------------------------------->|
|                                                |
|<-----------------------------------------------+
|                     2.05 CONTENT Observe: 42   |
|                      Payload: []               |
|                        .                       |
|                        .                       |
|                        .                       |
|                                                |
|          (Access Tokens t1 and t2 issued       |
|          and successfully submitted to RS)     |
|                        .                       |
|                        .                       |
|                        .                       |
|                                                |
|           (Access Token t1 is revoked)         |
|                                                |
|<-----------------------------------------------+
|            2.05 CONTENT Observe: 53            |
|             Payload: [                         |
|                        [ [], [bstr.h(t1)] ]    |
|                      ]                         |
|                        .                       |
|                        .                       |
|                        .                       |
|                                                |
|                                                |
|           (Access Token t2 is revoked)         |
|                                                |
|                                              |
|<---------------------------------------------+
|            2.05 CONTENT Observe: 64          |
|             Payload: [                       |
|                        [ [], [bstr.h(t2)] ], |
|                        [ [], [bstr.h(t1)] ]  |
|                      ]                       |
|                        .                     |
|                        .                     |
|                        .                     |
|                                              |
|          (Access Token t1 expires)           |
|                                              |
|<---------------------------------------------+
|            2.05 CONTENT Observe: 75          |
|             Payload: [                       |
|                        [ [bstr.h(t1)], [] ], |
|                        [ [], [bstr.h(t2)] ], |
|                        [ [], [bstr.h(t1)] ]  |
|                      ]                       |
|                        .                     |
|                        .                     |
|                        .                     |
|                                              |
|          (Access Token t2 expires)           |
|                                              |
|<---------------------------------------------+
|            2.05 CONTENT Observe: 86          |
|             Payload: [                       |
|                        [ [bstr.h(t2)], [] ], |
|                        [ [bstr.h(t1)], [] ], |
|                        [ [], [bstr.h(t2)] ]  |
|                      ]                       |
|                                              |
```

Figure 6: Interaction for Diff Query with Observation

8.3. Full Query with Observation and Additional Diff Query

Figure 7 shows an example interaction between the Resource Server RS and the Authorization Server AS, considering a CoAP observation and a full query of the TRL.

The example also considers one of the notifications from the Authorization Server to get lost in transmission, and thus not reaching the Resource Server.

   When this happens, and after a waiting time defined by the
   application has elapsed, the Resource Server sends a GET request with
   no observation to the Authorization Server, to perform a diff query
   of the TRL.

   In particular, the Resource Server indicates N=8 as value of the
   query parameter "diff", i.e. as the maximum number of diff entries to
   be specified in a response from the Authorization Server.

```
             RS                                            AS
             |                                              |
             | Registration: POST                           |
             +--------------------------------------------->|
             |                                              |
             |<---------------------------------------------+
             |                  2.01 CREATED                |
             |                   Payload: {                 |
             |                      ...                     |
             |                      "trl" = "revoke/trl",   |
             |                      "trl-hash" = "sha-256", |
             |                      "n-max" = 10            |
             |                   }                          |
             |                                              |
             | GET Observe: 0                               |
             |  coap://example.as.com/revoke/trl/           |
             +--------------------------------------------->|
             |                                              |
             |<---------------------------------------------+
             |                     2.05 CONTENT Observe: 42 |
             |                      Payload: []             |
             |                       .                      |
             |                       .                      |
             |                       .                      |
             |                                              |
             |      (Access Tokens t1 and t2 issued         |
             |      and successfully submitted to RS)       |
             |                       .                      |
             |                       .                      |
             |                       .                      |
             |                                              |
             |         (Access Token t1 is revoked)         |
             |                                              |
             |<---------------------------------------------+
             |                     2.05 CONTENT Observe: 53 |
             |                      Payload: [bstr.h(t1)]   |
             |                       .                      |
             |                       .                      |
             |                       .                      |
             |                                              |
             |         (Access Token t2 is revoked)         |
             |                                              |
             |<---------------------------------------------+
             |                     2.05 CONTENT Observe: 64 |
             |                      Payload: [bstr.h(t1),   |
             |                                bstr.h(t2)]   |
             |                       .                      |
             |                       .                      |
             |                       .                      |
             |                                              |
             |         (Access Token t1 expires)            |
             |                                              |
             |<---------------------------------------------+
             |                     2.05 CONTENT Observe: 75 |
             |                      Payload: [bstr.h(t2)]   |
             |                        .                     |
             |                        .                     |
             |                        .                     |
             |                                              |
             |         (Access Token t2 expires)            |
             |                                              |
             |       X<-------------------------------------+
             |                     2.05 CONTENT Observe: 86 |
             |                      Payload: []             |
             |                        .                     |
             |                        .                     |
             |                        .                     |
             |                                              |
             |       (Enough time has passed since          |
             |       the latest received notification)      |
             |                                              |
             | GET                                          |
             |  coap://example.as.com/revoke/trl?diff=8     |
             +--------------------------------------------->|
             |                                              |
             |<---------------------------------------------+
             |            2.05 CONTENT                      |
             |             Payload: [                       |
             |                        [ [bstr.h(t2)], [] ], |
             |                        [ [bstr.h(t1)], [] ], |
             |                        [ [], [bstr.h(t2)] ], |
             |                        [ [], [bstr.h(t1)] ]  |
             |                      ]                       |
             |                                              |
```

Figure 7: Interaction for Full Query with Observation and Diff Query

9. Security Considerations

Security considerations are inherited from the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], from [RFC8392] as to the usage of CWTs, from [RFC7519] as to the usage of JWTs, from [RFC7641] as to the usage of CoAP Observe, and from [RFC6920] with regards to resource naming through hashes. The following considerations also apply.

The Authorization Server MUST ensure that each registered device can access and retrieve only its pertaining portion of the TRL. To this end, the Authorization Server can perform the required filtering based on the authenticated identity of the registered device, i.e., a (non-public) identifier that the Authorization Server can securely relate to the registered device and the secure association they use to communicate.

Disclosing any information about revoked Access Tokens to entities other than the intended registered devices may result in privacy concerns. Therefore, the Authorization Server MUST ensure that, other than registered devices accessing their own pertaining portion of the TRL, only authorized and authenticated administrators can retrieve the full TRL. To this end, the Authorization Server may rely on an access control list or similar.

If a registered device has many non-expired Access Tokens associated to itself that are revoked, the pertaining portion of the TRL could grow to a size bigger than what the registered device is prepared to handle upon reception, especially if relying on a full query of the TRL resource (see Section 5.1). This could be exploited by attackers to negatively affect the behavior of a registered device. Short expiration times could help reduce the size of a TRL, but an Authorization Server SHOULD take measures to limit this size.

Most of the communication about revoked Access Tokens presented in this specification relies on CoAP Observe Notifications sent from the Authorization Server to a registered device. The suppression of those notifications by an external attacker that has access to the network would prevent registered devices from ever knowing that their pertaining Access Tokens have been revoked. To avoid this, a registered device SHOULD NOT rely solely on the CoAP Observe notifications. In particular, a registered device SHOULD also regularly poll the Authorization Server for the most current information about revoked Access Tokens, by sending GET requests to the TRL endpoint according to an application policy.

10. IANA Considerations

This document has no actions for IANA.

11. References

11.1. Normative References

[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-35 (work in progress), June 2020.

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work in progress), September 2020.

[Named.Information.Hash.Algorithm]
IANA, "Named Information Hash Algorithm", <https://www.iana.org/assignments/named-information/named-information.xhtml>.

[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>.

[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>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[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>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[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>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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>.

11.2. Informative References

[I-D.bormann-t2trg-stp] Bormann, C. and K. Hartke, "The Series Transfer Pattern (STP)", draft-bormann-t2trg-stp-03 (work in progress), April 2020.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

[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>.

Appendix A. Usage of the Series Transfer Pattern

This section discusses how the diff query of the TRL defined in Section 5.2 is a usage example of the Series Transfer Pattern defined in [I-D.bormann-t2trg-stp].

A diff query enables the transfer of a series of TRL updates, with the Authorization Server specifying U <= N_MAX diff entries as the U most recent updates to the portion of the TRL pertaining to a registered device.

For each registered device, the Authorization Server maintains an update collection of maximum N_MAX items. Each time the TRL changes, the Authorization Server performs the following operations for each registered device.

1. The Authorization Server considers the portion of the TRL pertaining to that registered device. If the TRL portion is not affected by this TRL update, the Authorization Server stops the processing for that registered device.

2. Otherwise, the Authorization Server creates two sets 'trl_patch' of token hashes, i.e. one "removed" set and one "added" set, as related to this TRL update.

3. The Authorization Server fills the two sets with the token hashes of the removed and added Access Tokens, respectively, from/to the TRL portion from step 1.

4. The Authorization Server creates a new series item including the two sets from step 3, and adds the series item to the update collection associated to the registered device.

When responding to a diff query request from a registered device (see Section 5.2), 'diff' is a subset of the collection associated to the requester, where each 'diff_entry' record is a series item from that collection. Note that 'diff' specifies the whole current collection when the value of U is equal to SIZE, i.e. the current number of series items in the collection.

The value N of the 'diff' query parameter in the diff query request allows the requester and the Authorization Server to trade the amount of provided information with the latency of the information transfer.

Since the collection associated to each registered device includes up to N_MAX series item, the Authorization Server deletes the oldest series item when a new one is generated and added to the end of the collection, due to a new TRL update pertaining to that registered device. This addresses the question "When can the server decide to no longer retain older items?" in Section 3.2 of [I-D.bormann-t2trg-stp].

Appendix B. Usage of the "Cursor" Pattern

Building on Appendix A, this section describes how the diff query of the TRL defined in Section 5.2 can be further improved by using the "Cursor" pattern of the Series Transfer Pattern (see Section 3.3 of [I-D.bormann-t2trg-stp]).

This has two benefits. First, the Authorization Server can avoid excessively big latencies when several diff entries have to be transferred, by delivering one adjacent subset at the time, in different diff query responses. Second, a requester can retrieve diff entries associated to TRL updates that, even if not the most recent ones, occurred after a TRL update indicated as checkpoint.

To this end, each series item in an update collection is also associated with an unsigned integer 'index', with value the absolute counter of series items added to that collection minus 1. That is, the first series item added to a collection has 'index' with value 0. Then, the values of 'index' are used as cursor information.

Furthermore, the Authorization Server defines an unsigned integer MAX_DIFF_BATCH <= N_MAX, specifying the maximum number of diff entries to be included in a single diff query response. If supporting diff queries, the Authorization Server should provide registered devices and administrators with the value of MAX_DIFF_BATCH, upon their registration.

Finally, the full query and diff query exchanges defined in Section 5.1 and Section 5.2 are extended as follows.

B.1. Full Query Request

No changes apply to what defined in Section 5.1.

B.2. Full Query Response

The response to a full query request (see Section 5.1) includes the CBOR array of token hashes as well as a parameter 'cursor', encoded either as a CBOR unsigned integer or the CBOR simple value Null.

The 'cursor' parameter specifies the value Null if there are currently no updates pertinent to the requester, i.e. the update collection for that requester is empty. This is the case from when the requester registers at the Authorization Server until a first update pertaining that requester occurs to the TRL.

Otherwise, the 'cursor' parameter takes the value of 'index' for the last series item in the collection, as corresponding to the most recent update pertaining to the requester occurred to the TRL.

B.3. Diff Query Request

In addition to the query parameter 'diff' (see Section 5.2), the requester can specify a query parameter 'cursor', with value an unsigned integer.

B.4. Diff Query Response

When receiving the diff query request, the Authorization Server proceeds as follows.

B.4.1. Empty Collection

If the collection associated to the requester has no elements, the Authorization Server returns a diff query response that contains:

- o The 'diff' parameter, encoding an empty CBOR array.
- o A 'cursor' parameter, encoding the CBOR simple value Null.
- o A 'more' parameter, encoding the CBOR simple value False.

B.4.2. Cursor Not Specified in the Diff Query Request

If the update collection associated to the requester is not empty and the diff query request does not include the query parameter 'cursor', the Authorization Server returns a diff query response that contains:

- o The 'diff' CBOR array, including L = min(U, MAX_DIFF_BATCH) diff entries. In particular:

  * If U <= MAX_DIFF_BATCH, these diff entries are the last series items in the collection associated to the requester, corresponding to the L most recent TRL updates pertaining to the requester.

  * If U > MAX_DIFF_BATCH, these diff entries are the eldest of the last L series items in the collection associated to the requester, as corresponding to the first L of the U most recent TRL updates pertaining to the requester.

- o A 'cursor' parameter, encoded as a CBOR unsigned integer. This takes the 'index' value of the series element of the collection included as first diff entry in the 'diff' CBOR array. That is, it takes the 'index' value of the series item in the collection corresponding to the most recent update pertaining to the requester and returned in this diff query response.

  Note that 'cursor' takes the same 'index' value of the last series item in the collection when U <= MAX_DIFF_BATCH.

- o A 'more' parameter, encoded as the CBOR simple value False if U <= MAX_DIFF_BATCH, or as the CBOR simple value True otherwise.

If 'more' has value True, the requester can send a follow-up diff query request including the query parameter 'cursor', with the same value of the 'cursor' parameter included in this response. This would result in the Authorization Server transfering the following subset of series items as diff entries, i.e. resuming from where interrupted in the previous transfer.

B.4.3. Cursor Specified in the Diff Query Request

If the update collection associated to the requester is not empty and the diff query request includes the query parameter 'cursor' with value P, the Authorization Server proceeds as follows.

- If no series item X with 'index' having value P is found in the collection associated to the requester, then that item has been previously removed from the history of updates for that requester (see Appendix A). In this case, the Authorization Server returns a diff query response that contains:

    * The 'diff' parameter, encoding an empty CBOR array.

    * A 'cursor' parameter, encoding the CBOR simple value Null.

    * A 'more' parameter, encoding the CBOR simple value True.

  With the combination ('cursor', 'more') = (Null, True), the Authorization Server is signaling that the update collection is in fact not empty, but that some series items have been lost due to their removal, including the item with 'index' value P that the requester wished to use as checkpoint.

  When receiving this diff query response, the requester should send a new full query request to the Authorization Server, in order to fully retrieve the current pertaining portion of the TRL.

- If the series item X with 'index' having value P is found in the collection associated to the requester, the Authorization Server returns a diff query response that contains:

    * The 'diff' CBOR array, including L = min(SUB_U, MAX_DIFF_BATCH) diff entries, where SUB_U = min(NUM, SUB_SIZE), and SUB_SIZE is the number of series items in the collection following the series item X.

      That is, these are the L updates pertaining to the requester that immediately follow the series item X indicated as checkpoint. In particular:

  + If SUB_U <= MAX_DIFF_BATCH, these diff entries are the last series items in the collection associated to the requester, corresponding to the L most recent TRL updates pertaining to the requester.

  + If SUB_U > MAX_DIFF_BATCH, these diff entries are the eldest of the last L series items in the collection associated to the requester, corresponding to the first L of the SUB_U most recent TRL updates pertaining to the requester.

* A 'cursor' parameter, encoded as a CBOR unsigned integer. If L is equal to 0, i.e. the series item X is the last one in the collection, 'cursor' takes the same 'index' value of the last series item in the collection. Otherwise, 'cursor' takes the 'index' value of the series element of the collection included as first diff entry in the 'diff' CBOR array. That is, it takes the 'index' value of the series item in the collection corresponding to the most recent update pertaining to the requester and returned in this diff query response.

  Note that 'cursor' takes the same 'index' value of the last series item in the collection when SUB_U <= MAX_DIFF_BATCH.

* A 'more' parameter, encoded as the CBOR simple value False if SUB_U <= MAX_DIFF_BATCH, or as the CBOR simple value True otherwise.

  If 'more' has value True, the requester can send a follow-up diff query request including the query parameter 'cursor', with the same value of the 'cursor' parameter specified in this diff query response. This would result in the Authorization Server transfering the following subset of series items as diff entries, i.e. resuming from where interrupted in the previous transfer.

Acknowledgments

The authors sincerely thank Carsten Bormann, Benjamin Kaduk, Jim Schaad, Goeran Selander and Travis Spencer for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Ludwig Seitz
Combitech
Djaeknegatan 31
Malmoe SE-21135 Malmoe
Sweden

Email: ludwig.seitz@combitech.se

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Sebastian Echeverria
CMU SEI
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
United States of America

Email: secheverria@sei.cmu.edu

Grace Lewis
CMU SEI
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
United States of America

Email: glewis@sei.cmu.edu