Network Working Group
Internet-Draft
Intended status: Informational
Expires: 30 January 2021

C. Bormann
Universität Bremen TZI

29 July 2020

An Authorization Information Format (AIF) for ACE
draft-ietf-ace-aif-00

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.

Table of Contents

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-12, 6 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-dtls-authorize-12.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

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

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

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

This Internet-Draft will expire on 6 May 2021.

Copyright Notice

Table of Contents

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.

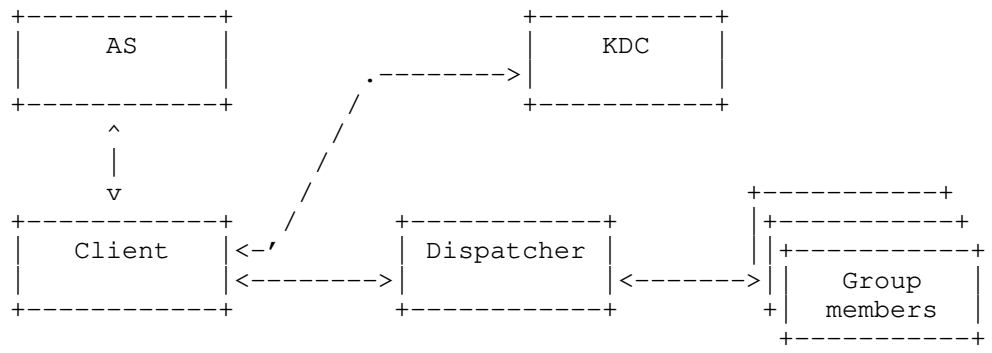


Figure 1: Key Distribution Participants

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.

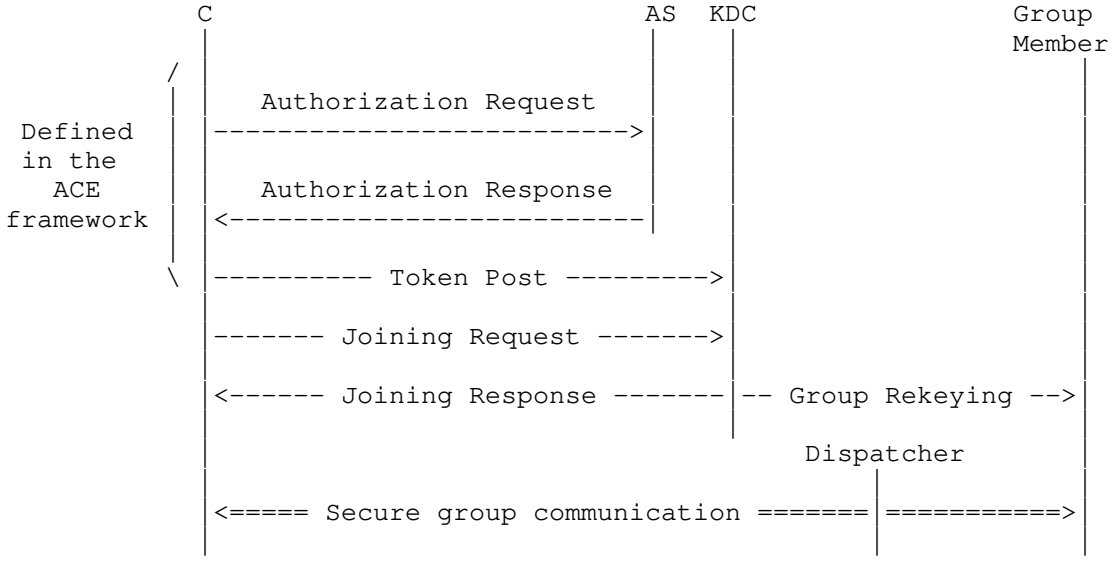


Figure 2: Message Flow Upon New Node's Joining

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.

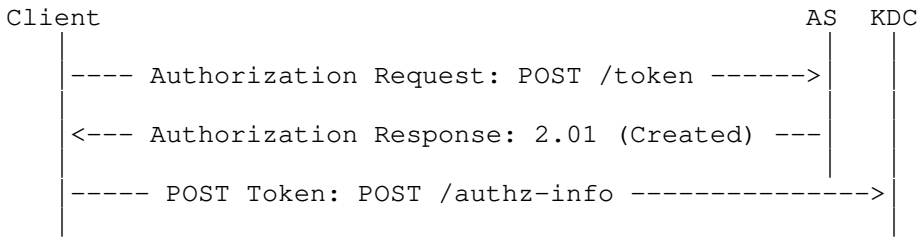


Figure 3: Message Flow of Join Authorization

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 Synchronization Method	TBD1	tstr/int	Method for a recipient 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.

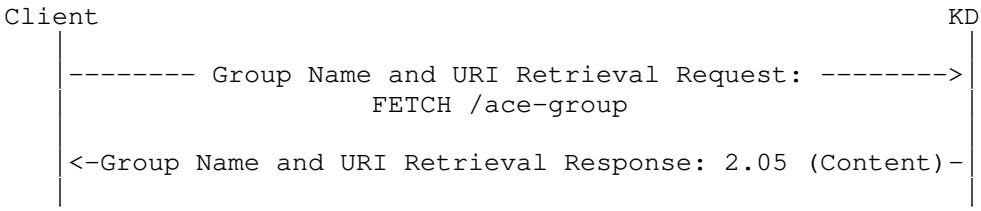


Figure 10: Message Flow of Group Name and URI Retrieval Request-Response

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.

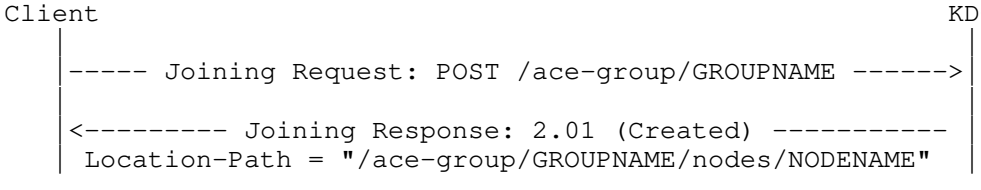


Figure 12: Message Flow of First Exchange for Group Joining

```
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.

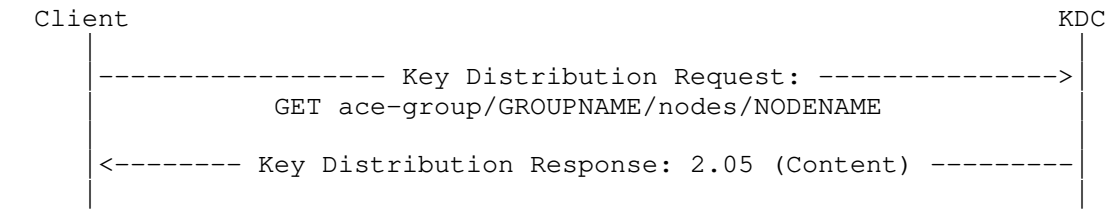


Figure 14: Message Flow of Key Distribution Request-Response

```
      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.

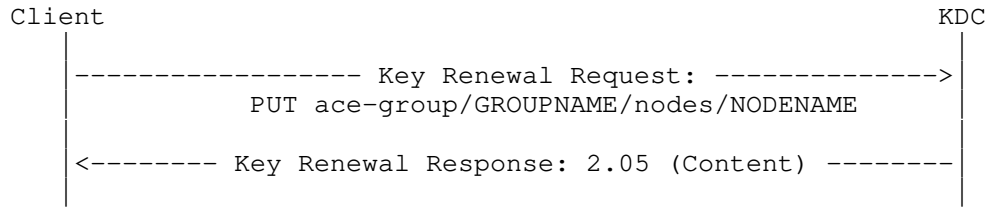


Figure 16: Message Flow of Key Renewal Request-Response

```
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.

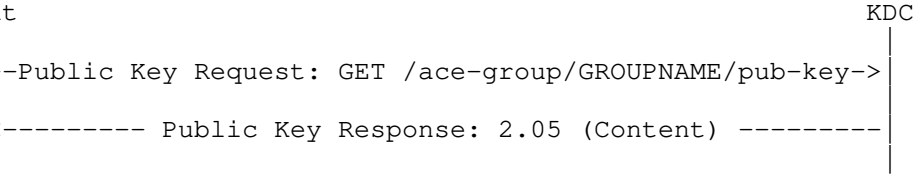


Figure 18: Message Flow of Public Key Exchange to Request All Members Public Keys

```
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

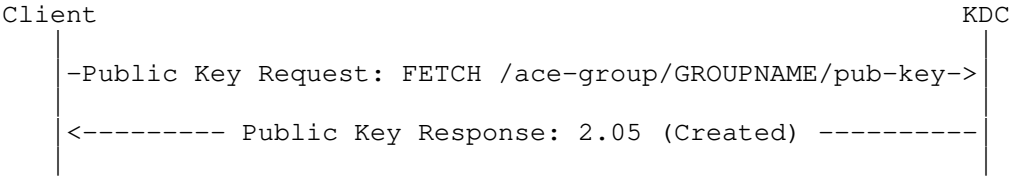


Figure 20: Message Flow of Public Key Exchange to Request Specific Members Public Keys

```
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.

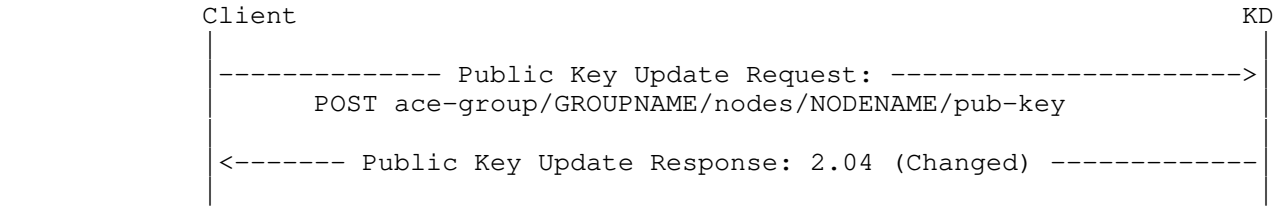


Figure 22: Message Flow of Public Key Update Request-Response

```
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.

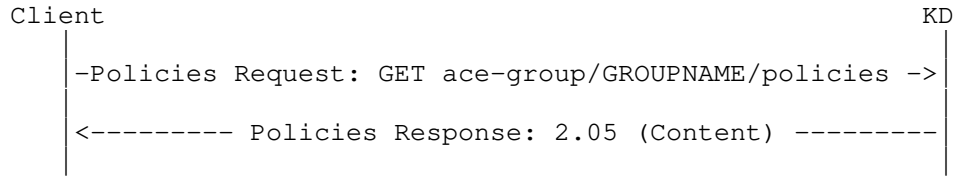


Figure 24: Message Flow of Policies Request-Response

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.

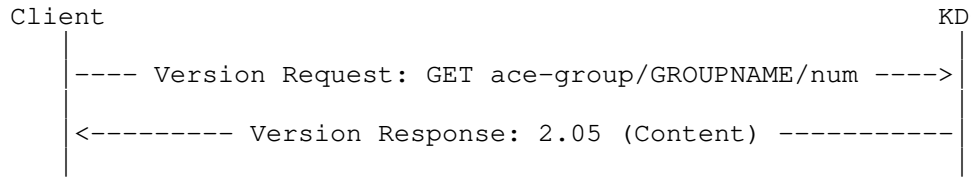


Figure 26: Message Flow of Version Request-Response

```
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
R. Hoeglund
RISE AB
P. van der Stok
Consultant
F. Palombini
K. Hartke
Ericsson AB

November 02, 2020

Admin Interface for the OSCORE Group Manager
draft-ietf-ace-oscore-gm-admin-01

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.

Table of Contents

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.ietf-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 [I-D.ietf-cbor-7049bis], 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 from the following specifications:

- CBOR [I-D.ietf-cbor-7049bis] and COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].

- The CoAP protocol [RFC7252], also in group communication scenarios [I-D.ietf-core-groupcomm-bis]. These include the concepts of:

  * "application group", as a set of CoAP nodes that share a common set of resources; and of

  * "security group", as a set of CoAP nodes that share the same security material, and use it to protect and verify exchanged messages.

- The OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm] security protocols. These include the concept of Group Manager, as the entity responsible for a set of OSCORE groups where communications among members are secured using Group OSCORE. An OSCORE group is used as security group for one or many application groups.

- 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).

- The management of keying material for groups in ACE [I-D.ietf-ace-key-groupcomm] and specifically for OSCORE groups [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.

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.

- 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 OSCORE 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 GROUPNAME as last segment, with GROUPNAME the invariant group name assigned upon its creation. Building on the considered url-path of the group-collection resource, this document uses /manage/ GROUPNAME 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 OSCORE group configurations, and the removal of OSCORE groups. Messages exchanged among the Administrator and the Group Manager are specified in Section 4.

2.2. Managing OSCORE Groups

Figure 1 shows the resources of a Group Manager available to an Administrator.

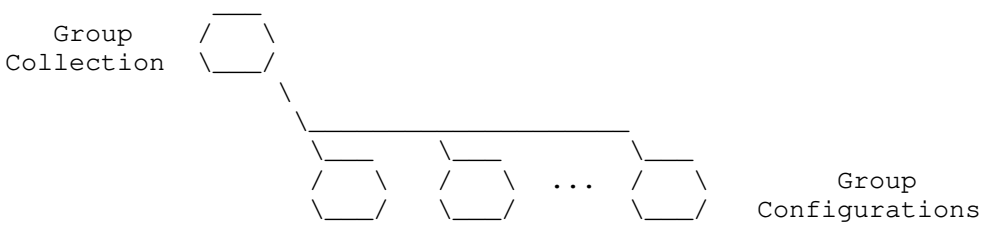


Figure 1: Resources of a Group Manager

The Group Manager exports a single group-collection resource, with resource type "core.osc.gcoll" defined in Section 6.2 of this specification. The interface for the group-collection resource defined in Section 4 allows the Administrator to:

- Retrieve the complete list of existing OSCORE groups.
- Retrieve a partial list of existing OSCORE groups, by applying filter criteria.
- 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 has resource type "core.osc.gconf" defined in Section 6.2 of this specification, and is identified by the group name specified upon creating the OSCORE group. The interface for a group-configuration resource defined in Section 4 allows the Administrator to:

- Retrieve the complete current configuration of the OSCORE group.
- Retrieve part of the current configuration of the OSCORE group, by applying filter criteria.
- Overwrite the current configuration of the OSCORE group.
- Delete the OSCORE group.

2.3. 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 former case, the link to each group-configuration resource specifies the link target attribute 'rt' (Resource Type), with value "core.osc.gconf" defined in Section 6.2 of this specification.

In the latter case, the CoRAL document specifies the group-configuration resources in the list as top-level elements. In particular, the link to each group-configuration resource has http://coreapps.org/core.osc.gcoll#item as relation type.

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 "core.osc.gcoll" defined in Section 6.2 of this specification.

The Administrator can discover group-configuration resources for the group-collection resource as specified in Section 4.1 and Section 4.2.

3. Group Configurations

A group configuration consists of a set of parameters.

3.1. Group Configuration Representation

The group configuration representation is a CBOR map which MUST include configuration properties and status properties.

3.1.1. Configuration Properties

The CBOR map MUST include the following configuration parameters:

- 'hkdf', defined in Section 6.1 of this document, specifies the HKDF algorithm used in the OSCORE group, encoded as a CBOR text string or a CBOR integer. 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].

- 'alg', defined in Section 6.1 of this document, specifies the AEAD algorithm used in the OSCORE group, encoded as a CBOR text string or a CBOR integer. 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].

- 'cs_alg', defined in Section 6.1 of this document, specifies the countersignature algorithm used in the OSCORE group, encoded as a CBOR text string or a CBOR 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].

- 'cs_params', defined in Section 6.1 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].

- 'cs_key_params', defined in Section 6.1 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 6.1 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].

- 'pairwise_mode', defined in Section 6.1 of this document and encoded as a CBOR simple value. Its value is True if the OSCORE group supports the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm], or False otherwise.

- 'ecdh_alg', defined in Section 6.1 of this document and formatted as follows. If the configuration parameter 'pairwise_mode' has value False, this parameter has as value the CBOR simple value Null. Otherwise, this parameter specifies the ECDH algorithm used in the OSCORE group, encoded as a CBOR text string or a CBOR integer. Possible values are the same ones admitted for the 'ecdh_alg' parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

- 'ecdh_params', defined in Section 6.1 of this document and formatted as follows. If the configuration parameter 'pairwise_mode' has value False, this parameter has as value the CBOR simple value Null. Otherwise, this parameter specifies the parameters for the ECDH algorithm used in the OSCORE group, encoded as a CBOR array. Possible formats and values are the same ones admitted for the 'ecdh_params' parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

- 'ecdh_key_params', defined in Section 6.1 of this document and formatted as follows. If the configuration parameter 'pairwise_mode' has value False, this parameter has as value the CBOR simple value Null. Otherwise, this parameter specifies the additional parameters for the key used with the ECDH algorithm in the OSCORE group, encoded as a CBOR array. Possible formats and values are the same ones admitted for the 'ecdh_key_params'

parameter of the "OSCORE Security Context Parameters" registry, defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore].

3.1.2. Status Properties

The CBOR map MUST include the following status parameters:

- 'rt', with value the resource type "core.osc.gconf" associated to group-configuration resources, encoded as a CBOR text string.

- 'active', encoding the CBOR simple value True if the OSCORE group is currently active, or the CBOR simple value False otherwise. This parameter is defined in Section 6.1 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 6.1 of this specification.

- 'group_title', with value either a human-readable description of the OSCORE group encoded as a CBOR text string, or the CBOR simple value Null if no description is specified. This parameter is defined in Section 6.1 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" defined in Section 21.3 of [I-D.ietf-ace-key-groupcomm-oscore] encoded as a CBOR integer.

- 'exp', defined in Section 4.1.2.1 of [I-D.ietf-ace-key-groupcomm].

- 'app_groups', with value a list of names of application groups, encoded as a CBOR array. Each element of the array is a CBOR text string, specifying the name of an application group using the OSCORE group as security group (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]).

- 'joining_uri', with value the URI of the group-membership resource for joining the newly created OSCORE group as per Section 6.2 of [I-D.ietf-ace-key-groupcomm-oscore], encoded as a CBOR text string. This parameter is defined in Section 6.1 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 6.1 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 Section 4 and Section 6 of [I-D.ietf-ace-key-groupcomm-oscore]).

3.2. Default Values

This section defines the default values that the Group Manager assumes for configuration and status parameters.

3.2.1. Configuration Parameters

For each configuration parameter, the Group Manager MUST use a pre-configured default value, if none is specified by the Administrator. In particular:

- For 'pairwise_mode', the Group Manager SHOULD use the CBOR simple value False.

- If 'pairwise_mode' has value True, the Group Manager SHOULD use the same default values defined in Section 19 of [I-D.ietf-ace-key-groupcomm-oscore] for the parameters 'ecdh_alg', 'ecdh_params' and 'ecdh_key_params'.

- For any other configuration parameter, the Group Manager SHOULD use the same default values defined in Section 19 of [I-D.ietf-ace-key-groupcomm-oscore].

3.2.2. Status Parameters

For the following status parameters, the Group Manager MUST use a pre-configured default value, if none is specified by the Administrator. In particular:

- For 'active', the Group Manager SHOULD use the CBOR simple value False.

- For 'group_title', the Group Manager SHOULD use the CBOR simple value Null.

- For 'app_groups', the Group Manager SHOULD use the empty CBOR array.

4. Interactions with the Group Manager

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 6.1 of this document MUST be used, when specifying the corresponding configuration and status parameters.

4.1. Retrieve the Full List of Groups Configurations

The Administrator can send a GET request to the group-collection resource, in order to retrieve the complete 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>;rt="core.osc.gconf",
   <coap://[2001:db8::ab]/manage/gp2>;rt="core.osc.gconf",
   <coap://[2001:db8::ab]/manage/gp3>;rt="core.osc.gconf"
```

Example in CoRAL:

```
=> 0.01 GET
   Uri-Path: manage

<= 2.05 Content
   Content-Format: TBD1 (application/coral+cbor)

   #using <http://coreapps.org/core.osc.gcoll#>
   #base </manage/>
   item <gp1>
   item <gp2>
   item <gp3>
```

4.2. Retrieve a List of 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 that fully match a set of specified filter criteria. This is returned as a list of links to the corresponding group-configuration resources.

When custom CBOR is used, the set of filter criteria is specified in the request payload as a CBOR map, whose possible entries are specified in Section 3.1 and use the same abbreviations defined in Section 6.1. Entry values are the ones admitted for the corresponding labels in the POST request for creating a group configuration (see Section 4.3). A valid request MUST NOT include the same entry multiple times.

When CoRAL is used, the filter criteria are specified in the request payload with top-level elements, each of which corresponds to an entry specified in Section 3.1, with the exception of the 'app_names' status parameter. If names of application groups are used as filter criteria, each element of the 'app_groups' array from the status properties is included as a separate element with name 'app_group'. With the exception of the 'app_group' element, a valid request MUST NOT include the same element multiple times. Element values are the ones admitted for the corresponding labels in the POST request for creating a group configuration (see Section 4.3).

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>;rt="core.osc.gconf",
   <coap://[2001:db8::ab]/manage/gp2>;rt="core.osc.gconf",
   <coap://[2001:db8::ab]/manage/gp3>;rt="core.osc.gconf"
```

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/core.osc.gcoll#>
      #base </manage/>
      item <gp1>
      item <gp2>
      item <gp3>
```

4.3. 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 GROUPNAME and group title, and MAY specify pieces of information concerning the group configuration.

When custom CBOR is used, the request payload is a CBOR map, whose possible entries are specified in Section 3.1 and use the same abbreviations defined in Section 6.1.

When CoRAL is used, each element of the request payload corresponds to an entry specified in Section 3.1, with the exception of the 'app_names' status parameter (see below).

In particular:

- The payload MAY include any of the configuration parameter defined in Section 3.1.1.

- The payload MAY include any of the status parameter 'group_name', 'group_title', 'exp', 'app_groups, 'group_policies', 'as_uri' and 'active' defined in Section 3.1.2.

  * When CoRAL is used, each element of the 'app_groups' array from the status properties is included as a separate element with name 'app_group'.

- The payload MUST NOT include any of the status parameter 'rt', 'ace-groupcomm-profile' and 'joining_uri' defined in Section 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.

- Any of the received parameters is specified multiple times, with the exception of the 'app_group' element when using CoRAL.

- Any of the received parameters is not recognized, or not valid, or not consistent with respect to other related parameters.

- 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/GROUPNAME, where GROUPNAME is the name of the OSCORE group as either indicated in the parameter 'group_name' of the request or uniquely assigned by the Group Manager. Note that the final decision about the name assigned to the OSCORE group is of the Group Manager, which may have more constraints than the Administrator can be aware of, possibly beyond the availability of suggested names.

The value of the status parameter 'rt' is set to "core.osc.gconf". The values of other parameters 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 use default values as specified in Section 3.2.

After that, the Group Manager creates a new group-membership resource accessible at ace-group/GROUPNAME to nodes that want to join the OSCORE group, as specified in Section 6.2 of [I-D.ietf-ace-key-groupcomm-oscore]. Note that such group membership-resource comprises a number of sub-resources intended to current group members, as defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm] and Section 5 of [I-D.ietf-ace-key-groupcomm-oscore].

From then on, the Group Manager will rely on the current group configuration to build the Joining Response message defined in Section 6.4 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 Group ID, used as OSCORE ID Context, 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 specifies the parameters 'group_name', 'joining_uri' and 'as_uri', from the status properties of the newly created OSCORE group (see Section 3.1), as detailed below.

When custom CBOR is used, the response payload is a CBOR map, where entries use the same abbreviations defined in Section 6.1. When CoRAL is used, the response payload includes one element for each specified parameter.

- 'group_name', with value the group name of the OSCORE group. 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. This parameter MUST be included.
- 'joining_uri', with value the URI of the group-membership resource for joining the newly created OSCORE group. This parameter MUST be included.
- 'as_uri', with value the URI of the Authorization Server associated to the Group Manager for the newly created OSCORE group. This parameter MUST be included if specified in the status properties of the group. 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.

The Group Manager can register the link to the group-membership resource with URI specified in 'joining_uri' to a Resource Directory [I-D.ietf-core-resource-directory][I-D.hartke-t2trg-coral-reef], as defined in Section 2 of [I-D.tiloca-core-oscore-discovery]. The

Group Manager considers the current group configuration when specifying additional information for the link to register.

Alternatively, the Administrator can perform the registration in 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.

- The names of the application groups using the OSCORE group MUST take the values possibly specified by the elements of the 'app_groups' parameter (when custom CBOR is used) or by the different 'app_group' elements (when CoRAL is used) in the POST request above.

- If present, parameters describing the cryptographic algorithms used in the OSCORE group MUST follow the values that the Administrator specified in the POST request above, or the corresponding default values specified in Section 3.2.1 otherwise.

- If also registering a related link to the Authorization Server associated to the OSCORE group, the related link MUST have as link target the URI in 'as_uri' from the 2.01 (Created) response above, if the 'as_uri' parameter was included in the response.

Note that, compared to the Group Manager, the Administrator is less likely to remain closely aligned with possible changes and updates that would require a prompt update to the registration in the Resource Directory. This applies especially to the address of the Group Manager, as well as the URI of the group-membership resource or of the Authorization Server associated to the Group Manager.

Therefore, it is RECOMMENDED that registrations of links to group-membership resources in the Resource Directory are made (and possibly updated) directly by the Group Manager, rather than by the Administrator.

Example in custom CBOR:

```
   => 0.02 POST
      Uri-Path: manage
      Content-Format: TBD2 (application/ace-groupcomm+cbor)

      {
        "alg" : 10,
        "hkdf" : 5,
        "pairwise_mode" : True,
        "active" : True,
        "group_title" : "rooms 1 and 2",
        "app_groups": : ["room1", "room2"],
        "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]/ace-group/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/core.osc.gconf#>
      alg 10
      hkdf 5
      pairwise_mode True
      active True
      group_title "rooms 1 and 2"
      app_group "room1"
      app_group "room2"
      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/core.osc.gconf#>
      group_name "gp4"
      joining_uri <coap://[2001:db8::ab]/ace-group/gp4/>
      as_uri <coap://as.example.com/token>
```

4.4. Retrieve a Group Configuration

The Administrator can send a GET request to the group-configuration resource manage/GROUPNAME associated to an OSCORE group with group name GROUPNAME, in order to retrieve the complete 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 3.1. The exact content of the payload reflects the current configuration of the OSCORE group. This includes both configuration properties and status properties.

When custom CBOR is used, the response payload is a CBOR map, whose possible entries are specified in Section 3.1 and use the same abbreviations defined in Section 6.1.

When CoRAL is used, the response payload includes one element for each entry specified in Section 3.1, with the exception of the 'app_names' status parameter. That is, each element of the 'app_groups' array from the status properties is included as a separate element with name 'app_group'.

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,
     "pairwise_mode" : True,
     "ecdh_alg" : -27,
     "ecdh_params" : [[1], [1, 6]],
     "ecdh_key_params" : [1, 6],
     "rt" : "core.osc.gconf",
     "active" : True,
     "group_name" : "gp4",
     "group_title" : "rooms 1 and 2",
     "ace-groupcomm-profile" : "coap_group_oscore_app",
     "exp" : "1360289224",
     "app_groups": : ["room1", "room2"],
     "joining_uri" : "coap://[2001:db8::ab]/ace-group/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/core.osc.gconf#>
      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
      pairwise_mode True
      ecdh_alg -27
      ecdh_params.alg_capab.key_type 1
      ecdh_params.key_type_capab.key_type 1
      ecdh_params.key_type_capab.curve 6
      ecdh_key_params.key_type 1
      ecdh_key_params.curve 6
      rt "core.osc.gconf",
      active True
      group_name "gp4"
      group_title "rooms 1 and 2"
      ace-groupcomm-profile "coap_group_oscore_app"
      exp "1360289224"
      app_group "room1"
      app_group "room2"
      joining_uri <coap://[2001:db8::ab]/ace-group/gp4/>
      as_uri <coap://as.example.com/token>
```

4.5. Retrieve Part of a Group Configuration by Filters

The Administrator can send a FETCH request to the group-configuration resource manage/GROUPNAME associated to an OSCORE group with group name GROUPNAME, in order to retrieve part of the current configuration of that group.

When custom CBOR is used, the request payload is a CBOR map, which contains the following fields:

- 'conf_filter', defined in Section 6.1 of this document and encoded as a CBOR array. Each element of the array specifies one requested configuration parameter or status parameter of the

current group configuration (see Section 3.1), using the corresponding abbreviation defined in Section 6.1.

When CoRAL is used, the request payload includes one element for each requested configuration parameter or status parameter of the current group configuration (see Section 3.1). All the specified elements have no value.

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 a partial representation of the group configuration (see Section 3.1). The exact content of the payload reflects the current configuration of the OSCORE group, and is limited to the configuration properties and status properties requested by the Administrator in the FETCH request.

The response payload includes the requested configuration parameters and status parameters, and is formatted as in the response payload of a GET request to a group-configuration resource (see Section 4.4).

Example in custom CBOR:

```
=> 0.05 FETCH
   Uri-Path: manage
   Uri-Path: gp4
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
     "conf_filter" : ["alg",
                      "hkdf",
                      "pairwise_mode",
                      "active",
                      "group_title",
                      "app_groups"]
   }

<= 2.05 Content
   Content-Format: TBD2 (application/ace-groupcomm+cbor)

   {
     "alg" : 10,
     "hkdf" : 5,
     "pairwise_mode" : True,
     "active" : True,
     "group_title" : "rooms 1 and 2",
     "app_groups": : ["room1", "room2"]
   }
```

Example in CoRAL:

```
=> 0.05 FETCH
   Uri-Path: manage
   Uri-Path: gp4
   Content-Format: TBD1 (application/coral+cbor)

   #using <http://coreapps.org/core.osc.gconf#>
   alg
   hkdf
   pairwise_mode
   active
   group_title
   app_groups

<= 2.05 Content
   Content-Format: TBD1 (application/coral+cbor)

   #using <http://coreapps.org/core.osc.gconf#>
   alg 10
   hkdf 5
   pairwise_mode True
   active True
   group_title "rooms 1 and 2"
   app_group "room1"
   app_group "room2"
```

4.6. Overwrite a Group Configuration

The Administrator can send a PUT request to the group-configuration resource associated to an OSCORE group, in order to overwrite the current configuration of that group with a new one. The payload of the request has the same format of the POST request defined in Section 4.3, 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 4.3. 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 parameter not specified in the PUT request, the Group Manager MUST use the default value as specified in Section 3.2. From then on, the Group Manager relies on the latest updated configuration to build the Joining Response message defined in Section 6.4 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 4.3.

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 in 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.

- The names of the application groups using the OSCORE group MUST take the values possibly specified by the elements of the 'app_groups' parameter (when custom CBOR is used) or by the different 'app_group' elements (when CoRAL is used) in the PUT request above.

- If present, parameters describing the cryptographic algorithms used in the OSCORE group MUST follow the values that the Administrator specified in the PUT request above, or the corresponding default values as specified in Section 3.2.1 otherwise.

- If also registering a related link to the Authorization Server associated to the OSCORE group, the related link MUST have as link target the URI in 'as_uri' from the 2.04 (Changed) response above, if the 'as_uri' parameter was included in the response.

As discussed in Section 4.3, it is RECOMMENDED that registrations of links to group-membership resources in the Resource Directory are made (and possibly updated) directly by the Group Manager, rather than by the Administrator.

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]/ace-group/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/core.osc.gconf#>
      alg 11
      hkdf 5

   <= 2.04 Changed
      Content-Format: TBD1 (application/coral+cbor)

      #using <http://coreapps.org/core.osc.gconf#>
      group_name "gp4"
      joining_uri <coap://[2001:db8::ab]/ace-group/gp4/>
      as_uri <coap://as.example.com/token>
```

4.6.1. 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 OSCORE group. In particular, upon receiving a Joining Request (see Section 6.3 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 OSCORE group, by processing their Joining Requests (see Section 6.3 of [I-D.ietf-ace-key-groupcomm-oscore]).

4.6.2. Effects on the Group Members

After having updated a group configuration, the Group Manager informs the members of the OSCORE group, over the pairwise secure communication channels established when joining the group (see Section 6 of [I-D.ietf-ace-key-groupcomm-oscore]).

To this end, the Group Manager can individually target the 'control_path' URI 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 OSCORE group (see Section 6.2 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 OSCORE group has been deactivated (i.e. 'active' has value False), SHOULD stop communicating in the group.

Every group member, upon learning that the OSCORE group has been reactivated (i.e. 'active' has value True again), can resume communicating in the group.

Every group member, upon learning that the OSCORE group has stopped supporting the pairwise mode of Group OSCORE (i.e. 'pairwise_mode' has value False), SHOULD stop using the pairwise mode to process messages in the group.

Every group member, upon learning that the OSCORE group has resumed supporting the pairwise mode of Group OSCORE (i.e. 'pairwise_mode' has value True again), can resume using the pairwise mode to process messages in the group.

Every group member, upon receiving updated values for 'alg' and 'hkdf', MUST either:

- Leave the OSCORE group (see Section 16 of [I-D.ietf-ace-key-groupcomm-oscore]), e.g. if not supporting the indicated new algorithms; or

- Use the new parameter values, and accordingly re-derive the OSCORE Security Context for the OSCORE 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', 'cs_key_enc', 'ecdh_alg', 'ecdh_params' and 'ecdh_key_params' MUST either:

- Leave the OSCORE group, e.g. if not supporting the indicated new algorithm, parameters and encoding; or

- Leave the OSCORE group and rejoin it (see Section 6 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

- Use the new parameter values, and, if required, provide the Group Manager with a new public key to use in the OSCORE group, as compatible with the indicated parameters (see Section 11 of [I-D.ietf-ace-key-groupcomm-oscore]).

4.7. Delete a Group Configuration

The Administrator can send a DELETE request to the group-configuration resource, in order to delete that OSCORE group. The deletion would be successful only on an inactive OSCORE group.

That is, the DELETE request actually yields a successful deletion of the OSCORE 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 OSCORE group (see Section 4.6), 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 associated to that group.

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
```

4.7.1. Effects on the Group Members

After having deleted an OSCORE 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 association established with that member during its joining process (see Section 6 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 18 of [I-D.ietf-ace-key-groupcomm-oscore]). The Group Manager uses the same format of the Joining Response message in Section 6.4 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 associated to the OSCORE 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 OSCORE group.

When being informed about the deletion of the OSCORE group, 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.

5. 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 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].

6. IANA Considerations

This document has the following actions for IANA.

6.1. 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	TBD	tstr / int	[[this document]]
alg	TBD	tstr / int	[[this document]]
cs_alg	TBD	tstr / int	[[this document]]
cs_params	TBD	array	[[this document]]
cs_key_params	TBD	array	[[this document]]
cs_key_enc	TBD	int	[[this document]]
pairwise_mode	TBD	simple value	[[this document]]
ecdh_alg	TBD	tstr / int / simple value	[[this document]]
ecdh_params	TBD	array / simple value	[[this document]]
ecdh_key_params	TBD	array / simple value	[[this document]]
active	TBD	simple value	[[this document]]
group_name	TBD	tstr	[[this document]]
group_title	TBD	tstr / simple value	[[this document]]
app_groups	TBD	array	[[this document]]
joining_uri	TBD	tstr	[[this document]]
as_uri	TBD	tstr	[[this document]]
conf_filter	TBD	array	[[this document]]

6.2. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values subregistry within the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690].

Value	Description	Reference
core.osc.gcoll	Group-collection resource of an OSCORE Group Manager	[[this document]]
core.osc.gconf	Group-configuration resource of an OSCORE Group Manager	[[this document]]

7. References

7.1. 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-10 (work in progress), November 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-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-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-coral]
Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-ietf-core-coral-03 (work in progress), March 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>.

[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>.

[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>.

7.2. Informative References

[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-14 (work in progress), October 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-25 (work in progress), July 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.

[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>.

Appendix A. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

A.1. Version -00 to -01

- Names of application groups as status parameter.
- Parameters related to the pairwise mode of Group OSCORE.
- Defined FETCH for group-configuration resources.
- Policies on registration of links to the Resource Directory.
- Added resource type for group-configuration resources.
- Fixes, clarifications and editorial improvements.

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; 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

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: June 17, 2021

F. Palombini
Ericsson AB
L. Seitz
Combitech
G. Selander
Ericsson AB
M. Gunnarsson
RISE

December 14, 2020

OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework
draft-ietf-ace-oscore-profile-14

Abstract

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

Table of Contents

1. Introduction

This memo specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use the Constrained Application Protocol (CoAP) [RFC7252] to communicate. The client uses an access token, bound to a symmetric key (the proof-of-possession key) to authorize its access to the resource server. Note that this profile uses a symmetric-crypto-based scheme, where the symmetric secret is used as input material for keying material derivation. In order to provide communication security and proof of possession, the client and resource server use Object Security for Constrained RESTful Environments (OSCORE) [RFC8613]. Note that the proof of possession is not achieved through a dedicated protocol element, but rather occurs after the first message exchange using OSCORE.

OSCORE specifies how to use CBOR Object Signing and Encryption (COSE) [RFC8152] to secure CoAP messages. Note that OSCORE can be used to secure CoAP messages, as well as HTTP and combinations of HTTP and CoAP; a profile of ACE similar to the one described in this document, with the difference of using HTTP instead of CoAP as communication protocol, could be specified analogously to this one.

1.1. Terminology

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

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify" are taken from [RFC4949].

RESTful terminology follows HTTP [RFC7231].

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

Concise Binary Object Representation (CBOR) [I-D.ietf-cbor-7049bis] and Concise Data Definition Language (CDDL) [RFC8610] are used in this specification. CDDL predefined type names, especially bstr for CBOR byte strings and tstr for CBOR text strings, are used extensively in the document.

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

2. Protocol Overview

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

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

This profile requires a client to retrieve an access token from the AS for the resource it wants to access on an RS, by sending an access token request to the token endpoint, as specified in section 5.6 of [I-D.ietf-ace-oauth-authz]. The access token request and response MUST be confidentiality-protected and ensure authenticity. This profile RECOMMENDS the use of OSCORE between client and AS, but other protocols (such as TLS or DTLS) can be used as well.

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

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

its OSCORE Sender ID), its own OSCORE Recipient ID, plus the parameters received in the access token from the AS, following section 3.2 of [RFC8613].

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

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

The use of random nonces during the exchange prevents the reuse of an Authenticated Encryption with Associated Data (AEAD) nonces/key pair for two different messages. Two-time pad might otherwise occur when client and RS derive a new Security Context from an existing (non-expired) access token, as might occur when either party has just rebooted. Instead, by using random nonces as part of the Master Salt, the request to the authz-info endpoint posting the same token results in a different Security Context, by OSCORE construction, since even though the Master Secret, Sender ID and Recipient ID are the same, the Master Salt is different (see Section 3.2.1 of [RFC8613]). Therefore, the main requirement for the nonces is that they have a good amount of randomness. If random nonces were not used, a node re-using a non-expired old token would be susceptible to on-path attackers provoking the creation of OSCORE messages using old AEAD keys and nonces.

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

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

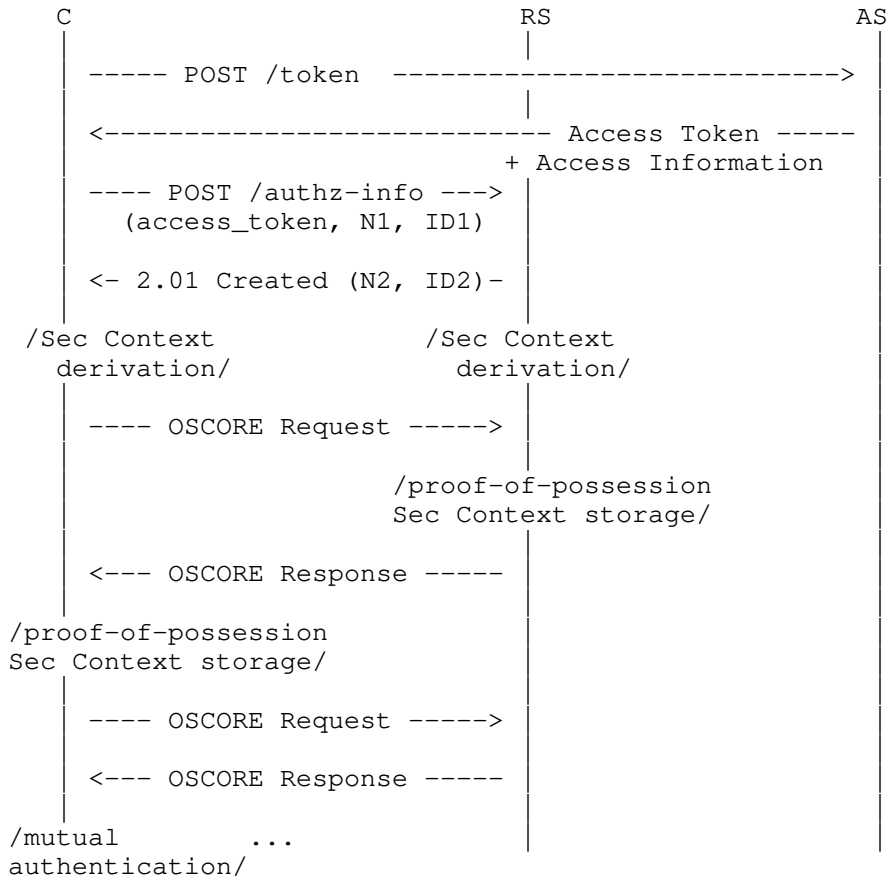


Figure 1: Protocol Overview

3. Client-AS Communication

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

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

The client-to-AS request is specified in Section 5.6.1 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 (see Section 5).

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:
{
  "req_aud" : "tempSensor4711",
  "scope" : "read"
 }
```

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

If the client wants to update its access rights without changing an existing OSCORE Security Context, it MUST include in its POST request to the token endpoint a req_cnf object, with the kid field carrying a CBOR byte string containing the OSCORE_Input_Material Identifier (assigned as discussed in Section 3.2). This identifier, together with other information such as audience (see Section 5.6.1 of [I-D.ietf-ace-oauth-authz]), can be used by the AS to determine the shared secret bound to the proof-of-possession token and therefore MUST identify a symmetric key that was previously generated by the AS as a shared secret for the communication between the client and the RS. The AS MUST verify that the received value identifies a proof-of-possession key that has previously been issued to the requesting client. If that is not the case, the Client-to-AS request MUST be declined with the error code 'invalid_request' as defined in Section 5.6.3 of [I-D.ietf-ace-oauth-authz].

An example of such a request, with payload in CBOR diagnostic notation without the tag and value abbreviations is reported in Figure 3

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

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

3.2. AS-to-C: Access Token

After verifying the POST request to the token endpoint and that the client is authorized to obtain an access token corresponding to its access token request, the AS responds as defined in section 5.6.2 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 is REQUIRED for a specific access token by including the "profile" parameter with the value "coap_oscore" in the access token response. This means that the client MUST use OSCORE towards all resource servers for which this access token is valid, and follow Section 4.3 to derive the security context to run OSCORE. Usually it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile signaling can be omitted.

Moreover, the AS MUST send the following data:

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

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

- a context identifier
- an AEAD algorithm
- an HMAC-based key derivation function (HKDF) algorithm

- a salt
- the OSCORE version number

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

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

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

The same parameters MUST be included in the claims associated with the access token. This profile RECOMMENDS the use of CBOR web token (CWT) as specified in [RFC8392]. If the token is a CWT, the same OSCORE_Input_Material structure defined above MUST be placed in the 'osc' field of the 'cnf' claim of this token.

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

Figure 4 shows an example of an AS response, 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 access token (CWT) omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600",
  "cnf" : {
    "osc" : {
      "id" : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f'
    }
  }
}
```

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

Figure 5 shows an example CWT Claims Set, including the relevant OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

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

Figure 5: Example CWT Claims Set with OSCORE parameters.

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

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

```
A5                                      # map(5)
   63                                   # text(3)
      617564                            # "aud"
   76                                   # text(22)
      74656D7053656E736F72496E4C6976696E67526F6F6D
                                        # "tempSensorInLivingRoom"
   63                                   # text(3)
      696174                            # "iat"
   6A                                   # text(10)
      31333630313839323234              # "1360189224"
   63                                   # text(3)
      657870                            # "exp"
   6A                                   # text(10)
      31333630323839323234              # "1360289224"
   65                                   # text(5)
      73636F7065                        # "scope"
   78 18                                # text(24)
      74656D7065726174757265F67206669726D776172655F70
                                        # "temperature_g firmware_p"
   63                                   # text(3)
      636E66                            # "cnf"
   A1                                   # map(1)
      63                                # text(3)
         6F7363                         # "osc"
      A2                                # map(2)
         62                             # text(2)
            6D73                        # "ms"
         50                             # bytes(16)
            F9AF838368E353E78888E1426BD94E6F
                                        # "\xF9\xAF\x83\x83h\xE3S\xE7
                                        \x88\x88\xE1Bk\xD9No"
         62                             # text(2)
            6964                        # "id"
         41                             # bytes(1)
            01                          # "\x01"
```

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

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

Figure 7 shows an example of such an AS response, 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 access token (CWT) omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600"
}
```

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

Figure 8 shows an example CWT Claims Set, containing the necessary OSCORE parameters in the 'cnf' claim for update of access rights, in CBOR diagnostic notation without tag and value abbreviations.

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

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

3.2.1. The OSCORE_Input_Material

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

(Section 9.4), defined for extensibility, and is specified below. All parameters are optional. Table 1 provides a summary of the OSCORE_Input_Material parameters defined in this section.

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

Table 1: OSCORE_Input_Material Parameters

id: This parameter identifies the OSCORE_Input_Material and is encoded as a byte string. In JSON, the "id" value is a Base64 encoded byte string. In CBOR, the "id" type is byte string, and has label 8.

version: This parameter identifies the OSCORE Version number, which is an unsigned integer. For more information about this field,

see section 5.4 of [RFC8613]. In JSON, the "version" value is an integer. In CBOR, the "version" type is integer, and has label 0.

ms: This parameter identifies the OSCORE Master Secret value, which is a byte string. For more information about this field, see section 3.1 of [RFC8613]. In JSON, the "ms" value is a Base64 encoded byte string. In CBOR, the "ms" type is byte string, and has label 1.

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

alg: This parameter identifies the OSCORE AEAD Algorithm. For more information about this field, see section 3.1 of [RFC8613] The values used MUST be registered in the IANA "COSE Algorithms" registry (see [COSE.Algorithms]) and MUST be AEAD algorithms. The value can either be the integer or the text string value of the HMAC-based HKDF algorithm in the "COSE Algorithms" registry. In JSON, the "alg" value is a case-sensitive ASCII string or an integer. In CBOR, the "alg" type is text string or integer, and has label 5.

salt: This parameter identifies an input to the OSCORE Master Salt value, which is a byte string. For more information about this field, see section 3.1 of [RFC8613]. In JSON, the "salt" value is a Base64 encoded byte string. In CBOR, the "salt" type is byte string, and has label 6.

contextId: This parameter identifies the security context as a byte string. This identifier is used as OSCORE ID Context. For more information about this field, see section 3.1 of [RFC8613]. In JSON, the "contextID" value is a Base64 encoded byte string. In CBOR, the "contextID" type is byte string, and has label 7.

An example of JSON OSCORE_Input_Material is given in Figure 9.

```
          "osc" : {
            "alg" : "AES-CCM-16-64-128",
            "id" : b64'AQ=='
            "ms" : b64'+a+Dg2jjU+eIiOFCa9lObw'
          }
```

Figure 9: Example JSON OSCORE_Input_Material

The CDDL grammar describing the CBOR OSCORE_Input_Material is:

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

4. Client-RS Communication

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

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

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

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

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

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

Note that the use of the payload and the Content-Format is different from what is described in section 5.8.1 of [I-D.ietf-ace-oauth-authz], which only transports the token without any CBOR wrapping. In this profile, the client MUST wrap the token, N1 and ID1 in a CBOR map. The client MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [I-D.ietf-ace-oauth-authz]. The client MUST include the access token using the 'access_token' parameter, N1 using the 'nonce1' parameter defined in Section 4.1.1, and ID1 using the 'ace_client_recipientid' parameter defined in Section 4.1.2.

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

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

The client may also chose to re-POST the access token in order to renew its OSCORE Security Context. In that case, the client and the RS will exchange newly generated nonces, re-negotiate identifiers,

and derive new keying material. The client and RS might decide to keep the same identifiers or renew them during the re-negotiation.

Figure 10 shows an example of the request sent from the client to the RS, with payload in CBOR diagnostic notation without the tag and value abbreviations. The access token has been truncated for readability.

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

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

If the client has already posted a valid token, has already established a security association with the RS, and wants to update its access rights, the client can do so by posting the new token (retrieved from the AS and containing the update of access rights) to the /authz-info endpoint. The client MUST protect the request using the OSCORE Security Context established during the first token exchange. The client MUST only send the 'access_token' field in the CBOR map in the payload, no nonce or identifier are sent. After proper verification (see Section 4.2), the RS will replace the old token with the new one, maintaining the same Security Context.

4.1.1. The Nonce 1 Parameter

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

4.1.2. The ace_client_recipientid Parameter

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

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

The RS MUST follow the procedures defined in section 5.8.1 of [I-D.ietf-ace-oauth-authz]: the RS must verify the validity of the token. If the token is valid, the RS must respond to the POST request with 2.01 (Created). If the token is valid but is associated to claims that the RS cannot process (e.g., an unknown scope), or if any of the expected parameters is missing (e.g., any of the mandatory parameters from the AS or the identifier 'id1'), or if any parameters received in the 'osc' field is unrecognized, the RS must respond with an error response code equivalent to the CoAP code 4.00 (Bad Request). In the latter two cases, the RS may provide additional information in the error response, in order to clarify what went wrong. The RS may make an introspection request (see Section 5.7.1 of [I-D.ietf-ace-oauth-authz]) to validate the token before responding to the POST request to the authz-info endpoint.

Additionally, the RS MUST generate a nonce N2 very unlikely to have been previously used with the same input keying material, and its own Recipient ID, ID2. The RS makes sure that ID2 does not collide with any of its Recipient IDs. The RS MUST ensure that ID2 is different from the value received in the ace_client_recipientid parameter. The RS sends N2 and ID2 within the 2.01 (Created) response. The payload of the 2.01 (Created) response MUST be a CBOR map containing the 'nonce2' parameter defined in Section 4.2.1, set to N2, and the 'ace_server_recipientid' parameter defined in Section 4.2.2, set to ID2. This profile RECOMMENDS to use a 64-bit long random number as nonce's value. The RS MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [I-D.ietf-ace-oauth-authz].

Figure 11 shows an example of the response sent from the RS to the client, with payload in CBOR diagnostic notation without the tag and value abbreviations.

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

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

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

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

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

4.2.1. The Nonce 2 Parameter

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

4.2.2. The ace_server_recipientid Parameter

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

4.3. OSCORE Setup

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

```
N1, N2 and input salt expressed in CBOR diagnostic notation:
      nonce1 = h'018a278f7faab55a'
      nonce2 = h'25a8991cd700ac01'
      input salt = h'f9af838368e353e78888e1426bd94e6f'

N1, N2 and input salt as CBOR encoded byte strings:
      nonce1 = 0x48018a278f7faab55a
      nonce2 = 0x4825a8991cd700ac01
      input salt = 0x50f9af838368e353e78888e1426bd94e6f

Master Salt = 0x50 f9af838368e353e78888e1426bd94e6f 48 018a278f7faab55a 48 25a899
1cd700ac01
```

Figure 12: Example of Master Salt construction using CBOR encoding

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

```
N1, N2 and input salt values:
      nonce1 = 0x018a278f7faab55a (8 bytes)
      nonce2 = 0x25a8991cd700ac01 (8 bytes)
      input salt = 0xf9af838368e353e78888e1426bd94e6f (16 bytes)

Input to Base64 encoding: 0x10 f9af838368e353e78888e1426bd94e6f 08 018a278f7faab5
5a 08 25a8991cd700ac01

Master Salt = b64'EPmvg4No41PniIjhQmvZTm8IAYonj3+qtVoIJaiZHNcArAE='
```

Figure 13: Example of Master Salt construction using Base64 encoding

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

If any of the expected parameters is missing (e.g., any of the mandatory parameters from the AS or the RS), or if ace_client_recipientid equals ace_server_recipientid, then the client MUST stop the exchange, and MUST NOT derive the Security Context. The client MAY restart the exchange, to get the correct security material.

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

After sending the 2.01 (Created) response, the RS MUST set the Master Salt of the Security Context created to communicate with the client to the concatenation of salt, N1, and N2, in the same way described above. An example of Master Salt construction using CBOR encoding is given in Figure 12 and using Base64 encoding is given in Figure 13. The RS MUST set the Sender ID from the ace_client_recipientid received in Section 4.1, and the Recipient ID from the ace_server_recipientid sent in Section 4.2. The RS MUST set the Master Secret from the parameter received from the AS and forwarded by the client in the access token in Section 4.1 after validation of the token as specified in Section 4.2. The RS MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version from the parameters

received from the AS and forwarded by the client in the access token in Section 4.1 after validation of the token as specified in Section 4.2, if present. In case an optional parameter is omitted, the default value SHALL be used as described in sections 3.2 and 5.4 of [RFC8613]. After that, the RS MUST derive the complete Security Context following section 3.2.1 of [RFC8613], and MUST associate this Security Context with the authorization information from the access token.

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

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

4.4. Access rights verification

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

5. Secure Communication with AS

As specified in the ACE framework (section 5.7 of [I-D.ietf-ace-oauth-authz]), the requesting entity (RS and/or client) and the AS communicates via the introspection or token endpoint. The

use of CoAP and OSCORE ([RFC8613]) for this communication is RECOMMENDED in this profile; other protocols (such as HTTP and DTLS or TLS) MAY be used instead.

If OSCORE is used, the requesting entity and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the requesting entity and the AS communicate through the introspection endpoint as specified in section 5.7 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

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

The client MUST discard the current Security Context associated with an RS when:

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

The RS MUST discard the current Security Context associated with a client when:

- the Sequence Number space ends.
- the access token associated with the context expires.
- the client has successfully replaced the current security context with a newer one by posting an access token to the unprotected /authz-info endpoint at the RS, e.g., by re-posting the same token, as specified in Section 4.1.

Whenever one more access token is successfully posted to the RS, and a new Security Context is derived between the client and RS, messages

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

7. Security Considerations

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

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

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

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

Since the use of nonces in the exchange guarantees uniqueness of AEAD keys and nonces, it is REQUIRED that nonces are not reused with the same input keying material even in case of re-boots. This document RECOMMENDS the use of 64 bit random nonces. Considering the birthday paradox, the average collision for each nonce will happen after 2^32 messages, which is considerably more token provisionings than expected for intended applications. If applications use something else, such as a counter, they need to guarantee that reboot and loss of state on either node does not provoke re-use. If that is not guaranteed, nodes are susceptible to re-use of AEAD (nonces, keys) pairs, especially since an on-path attacker can cause the client to use an arbitrary nonce for Security Context establishment by replaying client-to-server messages.

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

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

8. Privacy Considerations

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

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

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

The token is sent in the clear to the authz-info endpoint, so if a client uses the same single token from multiple locations with multiple Resource Servers, it can risk being tracked by the token's value even when the access token is encrypted.

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

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

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

9. IANA Considerations

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

9.1. ACE Profile Registry

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

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

9.2. OAuth Parameters Registry

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

- Parameter name: nonce1
- Parameter usage location: client-rs request
- Change Controller: IESG
- Specification Document(s): [[this specification]]

- Parameter name: nonce2
- Parameter usage location: rs-client response
- Change Controller: IESG
- Specification Document(s): [[this specification]]

- Parameter name: ace_client_recipientid
- Parameter usage location: client-rs request
- Change Controller: IESG
- Specification Document(s): [[this specification]]

- Parameter name: ace_server_recipientid
- Parameter usage location: rs-client response
- Change Controller: IESG
- Specification Document(s): [[this specification]]

9.3. OAuth Parameters CBOR Mappings Registry

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

- Name: nonce1
- CBOR Key: TBD1
- Value Type: bstr
- Reference: [[this specification]]

- Name: nonce2
- CBOR Key: TBD2
- Value Type: bstr
- Reference: [[this specification]]

- Name: ace_client_recipientid
- CBOR Key: TBD3
- Value Type: bstr
- Reference: [[this specification]]

- Name: ace_server_recipientid
- CBOR Key: TBD4
- Value Type: bstr
- Reference: [[this specification]]

9.4. OSCORE Security Context Parameters Registry

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

The columns of the registry are:

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

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

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

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

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

description This field contains a brief description for the field.

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

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

9.5. CWT Confirmation Methods Registry

The following registration is done for the CWT Confirmation Methods Registry following the procedure specified in section 7.2.1 of [RFC8747]:

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

9.6. JWT Confirmation Methods Registry

The following registration is done for the JWT Confirmation Methods Registry following the procedure specified in section 6.2.1 of [RFC7800]:

- Confirmation Method Value: "osc"
- Confirmation Method Description: OSCORE_Input_Material carrying the parameters for using OSCORE per-message security with implicit key confirmation
- Change Controller: IESG

- Specification Document(s): Section 3.2.1 of [[this specification]]

9.7. Expert Review Instructions

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

Expert reviewers should take into consideration the following points:

- Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments. Code points in other ranges should not be assigned for testing.
- Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

10. References

10.1. Normative References

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

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", 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-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-16 (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>.

[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>.

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

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

[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>.

10.2. Informative References

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

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

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

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

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

Appendix A. Profile Requirements

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

- Optionally define new methods for the client to discover the necessary permissions and AS for accessing a resource, different from the one proposed in: Not specified
- Optionally specify new grant types: Not specified
- Optionally define the use of client certificates as client credential type: Not specified
- Specify the communication protocol the client and RS the must use: CoAP
- Specify the security protocol the client and RS must use to protect their communication: OSCORE
- Specify how the client and the RS mutually authenticate: Implicitly by possession of a common OSCORE security context.

Note that the mutual authentication is not completed before the client has verified an OSCORE response using this security context.

- Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol: OSCORE algorithms; pre-established symmetric keys
- Specify a unique ace_profile identifier: coap_oscore
- If introspection is supported: Specify the communication and security protocol for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- Specify the communication and security protocol for interactions between client and AS: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- Specify how/if the authz-info endpoint is protected, including how error responses are protected: Not protected.
- Optionally define other methods of token transport than the authz-info endpoint: Not defined

Acknowledgments

The authors wish to thank Jim Schaad and Marco Tiloca for the input on this memo. Special thanks to the responsible area director Benjamin Kaduk for his extensive review and contributed text. Ludwig Seitz worked on this document as part of the CelticNext projects CyberWI, and CRITISEC with funding from Vinnova. The work on this document has been partly supported also by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
Combitech
Djaeknegatan 31
Malmoe 211 35
Sweden

Email: ludwig.seitz@combitech.se

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Martin Gunnarsson
RISE
Scheelevagen 17
Lund 22370
Sweden

Email: martin.gunnarsson@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 July 2021

F. Palombini
Ericsson
C. Sengul
Brunel University

3 January 2021

Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE) draft-ietf-ace-pubsub-profile-02

Abstract

This specification defines an application profile for authentication and authorization for publishers and subscribers in a pub-sub setting scenario in a constrained environment, using the ACE framework. This profile relies on transport layer or application layer security to authorize the publisher to the broker. Moreover, it relies on application layer security for publisher-broker and subscriber-broker communication.

Note to Readers

Source for this draft and an issue tracker can be found at https://github.com/ace-wg/pubsub-profile (https://github.com/ace-wg/pubsub-profile).

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 July 2021.

Copyright Notice

Table of Contents

1. Introduction

The publisher-subscriber setting allows for devices with limited reachability to communicate via a broker that enables store-and-forward messaging between the devices. The pub-sub scenario using the Constrained Application Protocol (CoAP) is specified in [I-D.ietf-core-coap-pubsub], while the one using MQTT is specified in REF MQTT. This document defines a way to authorize nodes in a CoAP

pub-sub type of setting, using the ACE framework [I-D.ietf-ace-oauth-authz], and to provide the keys for protecting the communication between these nodes. This document gives detailed specifications for MQTT and CoAP pub-sub, but can easily be adapted for other transport protocol as well.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz], [I-D.ietf-ace-key-groupcomm]. In particular, analogously to [I-D.ietf-ace-oauth-authz], terminology for entities in the architecture such as Client (C), Resource Server (RS), and Authorization Server (AS) is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors], and terminology for entities such as the Key Distribution Center (KDC) and Dispatcher in [I-D.ietf-ace-key-groupcomm].

Readers are expected to be familiar with terms and concepts of pub-sub group communication, as described in [I-D.ietf-core-coap-pubsub], or MQTT (REF MQTT pubsub).

2. Application Profile Overview

The objective of this document is to specify how to authorize nodes, provide keys, and protect a pub-sub communication, using [I-D.ietf-ace-key-groupcomm], which itself expands the Ace framework ([I-D.ietf-ace-oauth-authz]), and transport profiles ([I-D.ietf-ace-dtls-authorize], [I-D.ietf-ace-oscore-profile], REF MQTT profile). The pub-sub communication protocol can be based on CoAP, as described in [I-D.ietf-core-coap-pubsub], MQTT (see REF MQTT comm), or other transport.

The architecture of the scenario is shown in Figure 1.

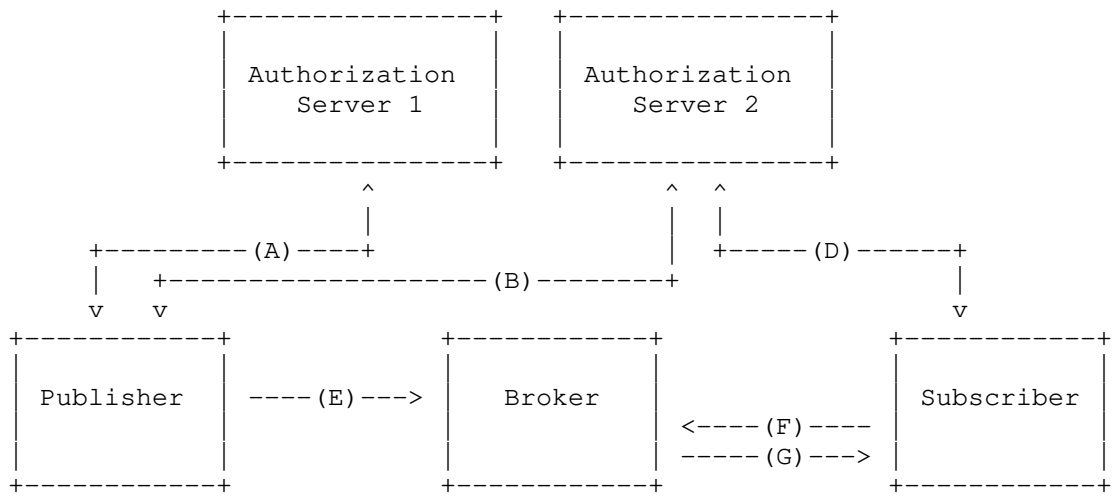


Figure 1: Architecture CoAP pubsub with Authorization Servers

The RS is the broker, which contains the topic. This node corresponds to the Dispatcher, in [I-D.ietf-ace-key-groupcomm]. The AS1 hosts the policies about the Broker: what endpoints are allowed to Publish on the Broker. The Clients access this node to get write access to the Broker. The AS2 hosts the policies about the topic: what endpoints are allowed to access what topic. This node represents both the AS and Key Distribution Center roles from [I-D.ietf-ace-key-groupcomm].

There are four phases, the first three can be done in parallel.

1. The Publisher requests publishing access to the Broker at the AS1, and communicates with the Broker to set up security.

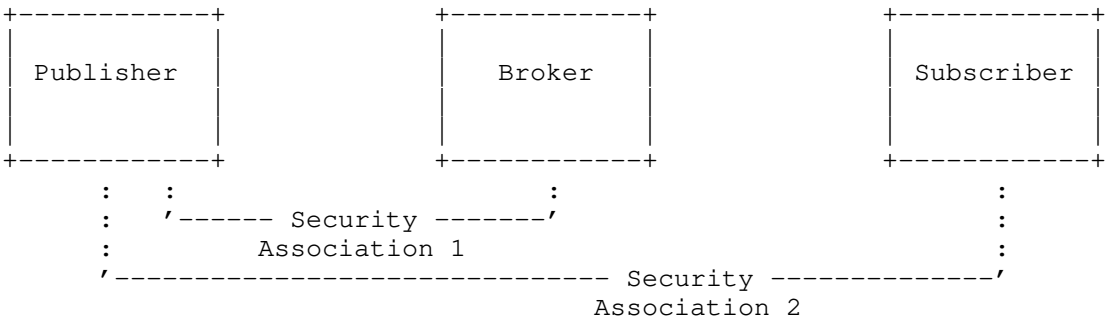
2. The Publisher requests access to a specific topic at the AS2

3. The Subscriber requests access to a specific topic at the AS2.

4. The Publisher and the Subscriber securely post to and get publications from the Broker.

This exchange aims at setting up 2 different security associations: on the one hand, the Publisher has a security association with the Broker, to protect the communication and securely authorize the Publisher to publish on a topic (Security Association 1). On the other hand, the Publisher has a security association with the Subscriber, to protect the publication content itself (Security Association 2). The Security Association 1 is set up using AS1 and a transport profile of [I-D.ietf-ace-oauth-authz], the Security Association 2 is set up using AS2 and [I-D.ietf-ace-key-groupcomm].

Note that, analogously to the Publisher, the Subscriber can also set up an additional security association with the Broker, using an AS, in the same way the Publisher does with AS1. In this case, only authorized Subscribers would be able to get notifications from the Broker. The overhead would be that each Subscriber should access the AS and get all the information to start a secure exchange with the Broker.



Note that AS1 and AS2 might either be co-resident or be 2 separate physical entities, in which case access control policies must be exchanged between AS1 and AS2, so that they agree on rights for joining nodes about specific topics. How the policies are exchanged is out of scope for this specification.

3. PubSub Application Profiles

Each profile defined in this document uses [I-D.ietf-ace-key-groupcomm], which expands the ACE framework. This section defines which exact parameters from [I-D.ietf-ace-key-groupcomm] have to be used, and the values for each parameter. Since [I-D.ietf-ace-oauth-authz] recommends the use of CoAP anc CBOR, this document describes the exchanges assuming CoAP and CBOR are used. However, using HTTP instead of CoAP is possible, using the corresponding parameters and methods. Analogously, JSON [RFC8259] can be used instead of CBOR, using the conversion method specified in Sections 4.1 and 4.2 of [RFC7049]. In case JSON is

used, the Content Format or Media Type of the message has to be changed accordingly.

The Publisher and the Subscriber map to the Client in [I-D.ietf-ace-key-groupcomm], the AS2 maps to the AS and to the KDC, the Broker maps to the Dispatcher.

Note that both publishers and subscribers use the same profile.

3.1. Retrieval of COSE Key for protection of content

This phase is common to both Publisher and Subscriber. To maintain the generality, the Publisher or Subscriber is referred as Client in this section.

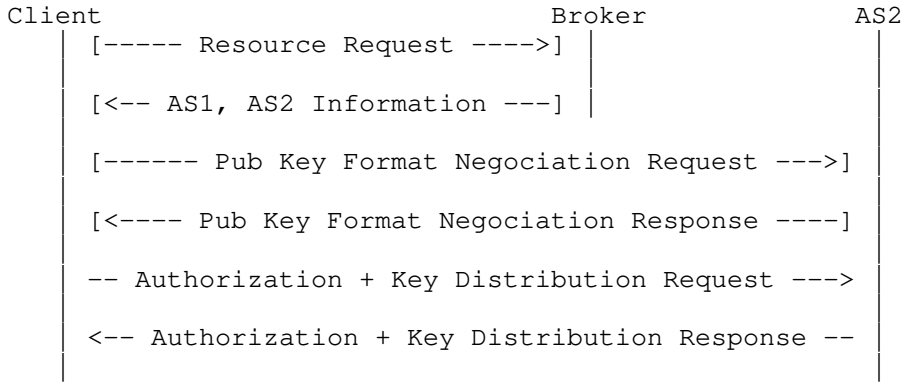


Figure 2: B: Access request - response

Complementary to what is defined in [I-D.ietf-ace-oauth-authz] (Section 5.1.1), to determine the AS2 in charge of a topic hosted at the Broker, the Broker MAY send the address of both the AS in charge of the topic back to the Client in the 'AS' parameter in the AS Information, as a response to an Unauthorized Resource Request (Section 5.1.2). The uri of AS2 is concatenated to the uri of AS1, and separated by a comma. An example using CBOR diagnostic notation and CoAP is given below:

```
4.01 Unauthorized
Content-Format: application/ace+cbor
{"AS": "coaps://as1.example.com/token,
coaps://as2.example.com/pubsubkey"}
```

Figure 3: AS1, AS2 Information example

After retrieving the AS2 address, the Client MAY send a request to the AS, in order to retrieve necessary information concerning the public keys in the group, as well as concerning the algorithm and related parameters for computing signatures in the group. This request is a subset of the Token POST request defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm], specifically a CoAP POST request to a specific resource at the AS, including only the parameters 'sign_info' and 'pub_key_enc' in the CBOR map in the payload. The default url-path for this resource is /ace-group/gid/cs-info, where "gid" is the topic identifier, but implementations are not required to use this name, and can use their own instead. The AS MUST respond with the response defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm], specifically including the parameters 'sign_info', 'pub_key_enc', and 'rsnonce' (8 bytes pseudo-random nonce generated by the AS).

After that, the Client sends an Authorization + Joining Request, which is an Authorization Request merged with a Joining Request, as described in [I-D.ietf-ace-key-groupcomm], Sections 3.1 and 4.2. The reason for merging these two messages is that the AS2 is both the AS and the KDC, in this setting, so the Authorization Response and the Post Token message are not necessary.

More specifically, the Client sends a POST request to the /ace-group/gid endpoint on AS2, with Content-Format = "application/ace+cbor" that MUST contain in the payload (formatted as a CBOR map):

* the following fields from the Joining Request (Section 4.2 of [I-D.ietf-ace-key-groupcomm]):

  - 'scope' parameter set to a CBOR array containing:

    o the broker's topic as first element, and

    o the text string "publisher" if the client request to be a publisher, "subscriber" if the client request to be a subscriber, or a CBOR array containing both, if the client request to be both.

  - 'get_pub_keys' parameter set to the empty array if the Client needs to retrieve the public keys of the other pubsub members,

  - 'client_cred' parameter containing the Client's public key formatted as a COSE_Key, if the Client needs to directly send that to the AS2,

  - 'cnonce', set to a 8 bytes long pseudo-random nonce, if 'client_cred' is present,

- 'client_cred_verify', set to a singature computed over the rsnonce concatenated with cnonce, if 'client_cred' is present,

- OPTIONALLY, if needed, the 'pub_keys_repos' parameter

* the following fields from the Authorization Request (Section 3.1 of [I-D.ietf-ace-key-groupcomm]):

- OPTIONALLY, if needed, additional parameters such as 'client_id'

TODO: 'cnonce' might change name. TODO: register media type ace+json for HTTP?

Note that the alg parameter in the 'client_cred' COSE_Key MUST be a signing algorithm, as defined in section 8 of [RFC8152], and that it is the same algorithm used to compute the signature sent in 'client_cred_verify'.

Examples of the payload of a Authorization + Joining Request are specified in Figure 5 and Figure 8.

The AS2 verifies that the Client is authorized to access the topic and, if the 'client_cred' parameter is present, stores the public key of the Client.

The AS2 response is an Authorization + Joining Response, with Content-Format = "application/ace+cbor". The payload (formatted as a CBOR map) MUST contain:

* the following fields from the Joining Response (Section 4.2 of [I-D.ietf-ace-key-groupcomm]):

- 'kty' identifies a key type "COSE_Key", as defined in Section 8.2.

- 'key', which contains a "COSE_Key" object (defined in [RFC8152], containing:

  o 'kty' with value 4 (symmetric)

  o 'alg' with value defined by the AS2 (Content Encryption Algorithm)

  o 'Base IV' with value defined by the AS2

  o 'k' with value the symmetric key value

- o OPTIONALLY, 'kid' with an identifier for the key value

- OPTIONALLY, 'exp' with the expiration time of the key

- 'pub_keys', containing the public keys of all authorized signing members formatted as COSE_Keys, if the 'get_pub_keys' parameter was present and set to the empty array in the Authorization + Key Distribution Request

* the following fields from the Authorization Response (Section 3.2 of [I-D.ietf-ace-key-groupcomm]):

  - 'profile' set to the corresponding value, see Section 3.2 or Section 3.3

  - OPTIONALLY 'scope', set to a CBOR array containing:

    - o the broker's topic as first element, and

    - o the string "publisher" if the client is an authorized publisher, "subscriber" if the client is an authorized subscriber, or a CBOR array containing both, if the client is authorized to be both.

Examples for the response payload are detailed in Figure 6 and Figure 9.

3.2. coap_pubsub_app Application Profile

In case CoAP PubSub is used as communication protocol:

* 'profile' set to "coap_pubsub_app", as specified in Section 8.1.1.

3.3. mqtt_pubsub_app Application Profile

In case mQTT PubSub is used as communication protocol:

* 'profile' set to "mqtt_pubsub_app", as specified in Section 8.1.2.

4. Publisher

In this section, it is specified how the Publisher requests, obtains and communicates to the Broker the access token, as well as the retrieval of the keying material to protect the publication.

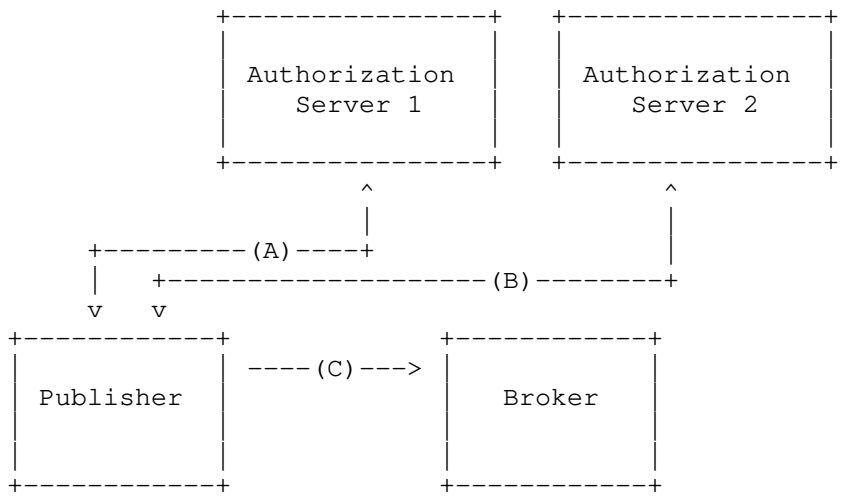


Figure 4: Phase 1: Publisher side

This is a combination of two independent phases:

* one is the establishment of a secure connection between Publisher and Broker, using an ACE transport profile such as DTLS [I-D.ietf-ace-dtls-authorize], OSCORE [I-D.ietf-ace-oscore-profile] or REF MQTT Profile. (A)(C)

* the other is the Publisher's retrieval of keying material to protect the publication. (B)

In detail:

(A) corresponds to the Access Token Request and Response between Publisher and Authorization Server to retrieve the Access Token and RS (Broker) Information. As specified, the Publisher has the role of a CoAP client, the Broker has the role of the CoAP server.

(C) corresponds to the exchange between Publisher and Broker, where the Publisher sends its access token to the Broker and establishes a secure connection with the Broker. Depending on the Information received in (A), this can be for example DTLS handshake, or other protocols. Depending on the application, there may not be the need for this set up phase: for example, if OSCORE is used directly. Note that, in line with what defined in the ACE transport profile used, the access token includes the scope (i.e. pubsub topics on the Broker) the Publisher is allowed to publish to. For implementation semplicity, it is RECOMMENDED that the ACE transport profile used and this specification use the same format of "scope".

(A) and (C) details are specified in the profile used.

(B) corresponds to the retrieval of the keying material to protect the publication end-to-end with the subscribers (see Section 6.1), and uses [I-D.ietf-ace-key-groupcomm]. The details are defined in Section 3.1.

4.1. CoAP Publisher

An example of the payload of an Authorization + Joining Request and corresponding Response for a CoAP Publisher using CoAP and CBOR is specified in Figure 5 and Figure 6, where SIG is a signature computed using the private key associated to the public key and the algorithm in "client_cred".

```
{
  "scope" : ["Broker1/Temp", "publisher"],
  "client_id" : "publisher1",
  "client_cred" :
    { / COSE_Key /
      / type / 1 : 2, / EC2 /
      / kid / 2 : h'11',
      / alg / 3 : -7, / ECDSA with SHA-256 /
      / crv / -1 : 1 , / P-256 /
      / x / -2 : h'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de1
      08de439c08551d',
      / y /-3 : h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e
      9eecd0084d19c',
  "cnonce" : h'd36b581d1eef9c7c,
  "client_cred_verify" : SIG
    }
}
```

Figure 5: Authorization + Joining Request payload for a Publisher

```
    {
      "profile" : "coap_pubsub_app",
      "kty" : "COSE_Key",
      "key" : {1: 4, 2: h'1234', 3: 12, 5: h'1f389d14d17dc7',
              -1: h'02e2cc3a9b92855220f255fff1c615bc'}
    }
```

Figure 6: Authorization + Joining Response payload for a Publisher

4.2. MQTT Publisher

TODO

5. Subscriber

In this section, it is specified how the Subscriber retrieves the keying material to protect the publication.

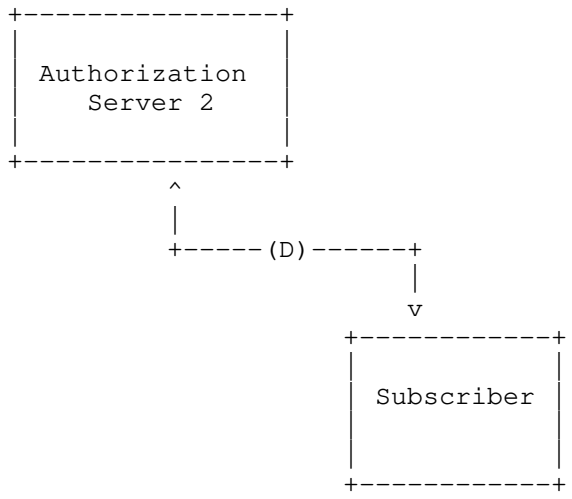


Figure 7: Phase 2: Subscriber side

Step (D) between Subscriber and AS2 corresponds to the retrieval of the keying material to verify the publication end-to-end with the publishers (see Section 6.1). The details are defined in Section 3.1

This step is the same as (B) between Publisher and AS2 (Section 3.1), with the following differences:

* The Authorization + Joining Request MUST NOT contain the 'client_cred parameter', the role element in the 'scope' parameter MUST be set to "subscriber". The Subscriber MUST have access to the public keys of all the Publishers; this MAY be achieved in the Authorization + Joining Request by using the parameter 'get_pub_keys' set to empty array.

* The Authorization + Key Distribution Response MUST contain the 'pub_keys' parameter.

5.1. CoAP Subscriber

An example of the payload of an Authorization + Joining Request and corresponding Response for a CoAP Subscriber using CoAP and CBOR is specified in Figure 8 and Figure 9.

```
{
  "scope" : ["Broker1/Temp", "subscriber"],
  "get_pub_keys" : [ ]
}
```

Figure 8: Authorization + Joining Request payload for a Subscriber

```
{
  "profile" : "coap_pubsub_app",
  "scope" : ["Broker1/Temp", "subscriber"],
  "kty" : "COSE_Key"
  "key" : {1: 4, 2: h'1234', 3: 12, 5: h'1f389d14d17dc7',
          -1: h'02e2cc3a9b92855220f255fff1c615bc'},
  "pub_keys" : [
   {
      1 : 2, / type EC2 /
      2 : h'11', / kid /
      3 : -7, / alg ECDSA with SHA-256 /
      -1 : 1 , / crv P-256 /
      -2 : h'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de43
      9c08551d', / x /
      -3 : h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd
      0084d19c' / y /
    }
  ]
}
```

Figure 9: Authorization + Joining Response payload for a Subscriber

5.2. MQTT Subscriber

TODO

6. Pub-Sub Protected Communication

This section specifies the communication Publisher-Broker and Subscriber-Broker, after the previous phases have taken place. The operations of publishing and subscribing are defined in [I-D.ietf-core-coap-pubsub].

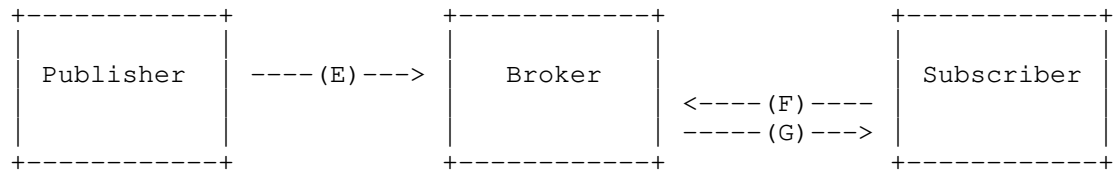


Figure 10: Phase 3: Secure communication between Publisher and Subscriber

The (E) message corresponds to the publication of a topic on the Broker. The publication (the resource representation) is protected with COSE ([RFC8152]). The (F) message is the subscription of the Subscriber, which is unprotected, unless a profile of ACE [I-D.ietf-ace-oauth-authz] is used between Subscriber and Broker. The (G) message is the response from the Broker, where the publication is protected with COSE.

The flow graph is presented below.

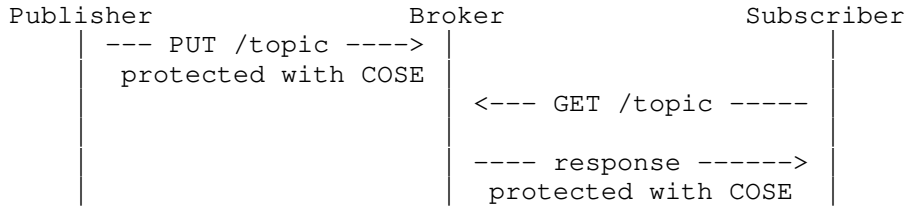


Figure 11: (E), (F), (G): Example of protected communication

6.1. Using COSE Objects To Protect The Resource Representation

The Publisher uses the symmetric COSE Key received from AS2 in exchange B (Section 3.1) to protect the payload of the PUBLISH operation (Section 4.3 of [I-D.ietf-core-coap-pubsub] and REF MQTT). Specifically, the COSE Key is used to create a COSE_Encrypt0 with algorithm specified by AS2. The Publisher uses the private key corresponding to the public key sent to the AS2 in exchange B (Section 3.1) to countersign the COSE Object as specified in Section 4.5 of [RFC8152]. The CoAP payload is replaced by the COSE object before the publication is sent to the Broker.

The Subscriber uses the kid in the countersignature field in the COSE object to retrieve the right public key to verify the countersignature. It then uses the symmetric key received from AS2 to verify and decrypt the publication received in the payload of the CoAP Notification from the Broker.

The COSE object is constructed in the following way:

* The protected Headers (as described in Section 3 of [RFC8152]) MAY contain the kid parameter, with value the kid of the symmetric COSE Key received in Section 3.1 and MUST contain the content encryption algorithm.

* The unprotected Headers MUST contain the Partial IV, with value a sequence number that is incremented for every message sent, and the counter signature that includes:

  - the algorithm (same value as in the asymmetric COSE Key received in (B)) in the protected header;

  - the kid (same value as the kid of the asymmetric COSE Key received in (B)) in the unprotected header;

  - the signature computed as specified in Section 4.5 of [RFC8152].

* The ciphertext, computed over the plaintext that MUST contain the CoAP payload.

The external_aad is an empty string.

An example is given in Figure 12

```
16(
  [
    / protected / h'a2010c04421234' / {
        \ alg \ 1:12, \ AES-CCM-64-64-128 \
        \ kid \ 4: h'1234'
      } / ,
    / unprotected / {
      / iv / 5:h'89f52f65a1c580',
      / countersign / 7:[
        / protected / h'a10126' / {
          \ alg \ 1:-7
        } / ,
        / unprotected / {
          / kid / 4:h'11'
        },
        / signature / SIG / 64 bytes signature /
      ]
    },
    / ciphertext / h'8df0a3b62fccff37aa313c8020e971f8aC8d'
  ]
)
```

Figure 12: Example of COSE Object sent in the payload of a PUBLISH operation

The encryption and decryption operations are described in sections 5.3 and 5.4 of [RFC8152].

7. Security Considerations

In the profile described above, the Publisher and Subscriber use asymmetric crypto, which would make the message exchange quite heavy for small constrained devices. Moreover, all Subscribers must be able to access the public keys of all the Publishers to a specific topic to be able to verify the publications. Such a database could be set up and managed by the same entity having control of the topic, i.e. AS2.

An application where it is not critical that only authorized Publishers can publish on a topic may decide not to make use of the asymmetric crypto and only use symmetric encryption/MAC to confidentiality and integrity protect the publication, but this is not recommended since, as a result, any authorized Subscribers with access to the Broker may forge unauthorized publications without being detected. In this symmetric case the Subscribers would only need one symmetric key per topic, and would not need to know any information about the Publishers, that can be anonymous to it and the Broker.

Subscribers can be excluded from future publications through re-keying for a certain topic. This could be set up to happen on a regular basis, for certain applications. How this could be done is out of scope for this work.

The Broker is only trusted with verifying that the Publisher is authorized to publish, but is not trusted with the publications itself, which it cannot read nor modify. In this setting, caching of publications on the Broker is still allowed.

TODO: expand on security and privacy considerations

8. IANA Considerations

8.1. ACE Groupcomm Profile Registry

The following registrations are done for the "ACE Groupcomm Profile" Registry following the procedure specified in [I-D.ietf-ace-key-groupcomm].

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

8.1.1. CoAP Profile Registration

Name: coap_pubsub_app

Description: Profile for delegating client authentication and authorization for publishers and subscribers in a CoAP pub-sub setting scenario in a constrained environment.

CBOR Key: TBD

Reference: [[This document]]

8.1.2. CoAP Profile Registration

Name: mqtt_pubsub_app

Description: Profile for delegating client authentication and authorization for publishers and subscribers in a MQTT pub-sub setting scenario in a constrained environment.

CBOR Key: TBD

Reference: [[This document]]

8.2. ACE Groupcomm Key Registry

The following registrations are done for the ACE Groupcomm Key Registry following the procedure specified in [I-D.ietf-ace-key-groupcomm].

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

Name: COSE_Key

Key Type Value: TBD

Profile: coap_pubsub_app

Description: COSE_Key object

References: [RFC8152], [[This document]]

9. References

9.1. Normative References

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-10, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-key-groupcomm-10.txt>.

[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-36, 16 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-36.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>.

[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>.

[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>.

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

9.2. Informative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", Work in Progress, Internet-Draft, draft-ietf-ace-actors-07, 22 October 2018, <http://www.ietf.org/internet-drafts/draft-ietf-ace-actors-07.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-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-14, 14 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oscore-profile-14.txt>.

[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>.

Appendix A. Requirements on Application Profiles

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

* REQ1: Specify the encoding and value of the identifier of group or topic of 'scope': see Section 3.1).

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

* REQ3: Optionally, specify the acceptable values for 'sign_alg': TODO

* REQ4: Optionally, specify the acceptable values for 'sign_parameters': TODO

* REQ5: Optionally, specify the acceptable values for 'sign_key_parameters': TODO

* REQ6: Optionally, specify the acceptable values for 'pub_key_enc': TODO

* REQ7: Specify the exact format of the 'key' value: COSE_Key, see Section 3.1.

* REQ8: Specify the acceptable values of 'kty' : "COSE_Key", see Section 3.1.

* REQ9: Specity the format of the identifiers of group members: TODO

* REQ10: Optionally, specify the format and content of 'group_policies' entries: not defined

* REQ11: Specify the communication protocol the members of the group must use: CoAP pub/sub.

* REQ12: Specify the security protocol the group members must use to protect their communication. This must provide encryption, integrity and replay protection: Object Security of Content using COSE, see Section 6.1.

* REQ13: Specify and register the application profile identifier : "coap_pubsub_app", see Section 8.1.

* REQ14: Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: NA.

* REQ15: Specify policies at the KDC to handle id that are not included in get_pub_keys: TODO

* REQ16: Specify the format and content of 'group_policies': TODO

* REQ17: Specify the format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label : not defined

* REQ18: 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: pre-set, as KDC is AS.

* OPT1: Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: NA

* OPT2: Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' and 'pub_key_enc' are not used: NA

* OPT3: Optionally, specify the format and content of 'mgt_key_material': not defined

* OPT4: Optionally, 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: not defined

Acknowledgments

The author wishes to thank Ari Keraenen, John Mattsson, Ludwig Seitz, Goeran Selander, Cigdem Sengul, Jim Schaad and Marco Tiloca for the useful discussion and reviews that helped shape this document.

Authors' Addresses

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

Cigdem Sengul
Brunel University

Email: csengul@acm.org