

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 26 June 2022

F. Palombini
Ericsson AB
M. Tiloca
RISE AB
23 December 2021

Key Provisioning for Group Communication using ACE
draft-ietf-ace-key-groupcomm-15

Abstract

This document defines how to use the Authentication and Authorization for Constrained Environments (ACE) framework to distribute keying material and configuration parameters for secure group communication. Candidate group members acting as Clients and authorized to join a group can do so by interacting with a Key Distribution Center (KDC) acting as Resource Server, from which they obtain the keying material to communicate with other group members. While defining general message formats as well as the interface and operations available at the KDC, this document supports different approaches and protocols for secure group communication. Therefore, details are delegated to separate application profiles of this document, as specialized instances that target a particular group communication approach and define how communications in the group are protected. Compliance requirements for such application profiles are also specified.

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 26 June 2022.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Overview	7
3. Authorization to Join a Group	10
3.1. Authorization Request	11
3.2. Authorization Response	13
3.3. Token Transferring	15
3.3.1. 'sign_info' Parameter	17
3.3.2. 'kdcchallenge' Parameter	19
4. KDC Functionalities	19
4.1. Interface at the KDC	20
4.1.1. Operations Supported by Clients	23
4.1.2. Error Handling	24
4.2. /ace-group	26
4.2.1. FETCH Handler	26
4.2.1.1. Retrieve Group Names	27
4.3. /ace-group/GROUPNAME	28
4.3.1. POST Handler	28
4.3.1.1. Join the Group	41
4.3.2. GET Handler	43
4.3.2.1. Retrieve Group Keying Material	44
4.4. /ace-group/GROUPNAME/pub-key	45
4.4.1. FETCH Handler	45
4.4.1.1. Retrieve a Subset of Public Keys in the Group	47
4.4.2. GET Handler	48
4.4.2.1. Retrieve All Public Keys in the Group	48
4.5. /ace-group/GROUPNAME/kdc-pub-key	49
4.5.1. GET Handler	49
4.5.1.1. Retrieve the KDC's Public Key	50
4.6. /ace-group/GROUPNAME/policies	51

4.6.1.	GET Handler	51
4.6.1.1.	Retrieve the Group Policies	52
4.7.	/ace-group/GROUPNAME/num	53
4.7.1.	GET Handler	53
4.7.1.1.	Retrieve the Keying Material Version	54
4.8.	/ace-group/GROUPNAME/nodes/NODENAME	54
4.8.1.	GET Handler	55
4.8.1.1.	Retrieve Group and Individual Keying Material	56
4.8.2.	PUT Handler	57
4.8.2.1.	Request to Change Individual Keying Material	59
4.8.3.	DELETE Handler	60
4.8.3.1.	Leave the Group	60
4.9.	/ace-group/GROUPNAME/nodes/NODENAME/pub-key	61
4.9.1.	POST Handler	61
4.9.1.1.	Uploading a New Public Key	62
5.	Removal of a Group Member	63
6.	Group Rekeying Process	65
6.1.	Point-to-Point Group Rekeying	66
6.2.	One-to-Many Group Rekeying	67
6.2.1.	Protection of Rekeying Messages	69
7.	Extended Scope Format	72
8.	ACE Groupcomm Parameters	74
9.	ACE Groupcomm Error Identifiers	77
10.	Security Considerations	79
10.1.	Secure Communication in the Group	79
10.2.	Update of Group Keying Material	80
10.2.1.	Misalignment of Group Keying Material	82
10.3.	Block-Wise Considerations	83
11.	IANA Considerations	83
11.1.	Media Type Registrations	83
11.2.	CoAP Content-Formats	84
11.3.	OAuth Parameters	84
11.4.	OAuth Parameters CBOR Mappings	85
11.5.	Interface Description (if=) Link Target Attribute Values	85
11.6.	CBOR Tags	86
11.7.	ACE Groupcomm Parameters	86
11.8.	ACE Groupcomm Key Types	87
11.9.	ACE Groupcomm Profiles	87
11.10.	ACE Groupcomm Policies	88
11.11.	Sequence Number Synchronization Methods	89
11.12.	ACE Scope Semantics	89
11.13.	ACE Groupcomm Errors	90
11.14.	ACE Groupcomm Rekeying Schemes	90
11.15.	Expert Review Instructions	91
12.	References	92
12.1.	Normative References	92
12.2.	Informative References	94

Appendix A. Requirements on Application Profiles	96
A.1. Mandatory-to-Address Requirements	96
A.2. Optional-to-Address Requirements	99
Appendix B. Extensibility for Future COSE Algorithms	100
B.1. Format of 'sign_info_entry'	100
Appendix C. Document Updates	101
C.1. Version -14 to -15	101
C.2. Version -13 to -14	101
C.3. Version -05 to -13	102
C.4. Version -04 to -05	102
C.5. Version -03 to -04	103
C.6. Version -02 to -03	103
C.7. Version -01 to -02	104
C.8. Version -00 to -01	105
Acknowledgments	105
Authors' Addresses	106

1. Introduction

This document builds on the Authentication and Authorization for Constrained Environments (ACE) framework and defines how to request, distribute and renew keying material and configuration parameters to protect message exchanges in a group communication environment.

Candidate group members acting as Clients and authorized to join a group can interact with the Key Distribution Center (KDC) acting as Resource Server and responsible for that group, in order to obtain the necessary keying material and parameters to communicate with other group members.

In particular, this document defines the operations and interface available at the KDC, as well as general message formats for the interactions between Clients and KDC. At the same time, communications in the group can rely on different approaches, e.g., based on multicast [I-D.ietf-core-groupcomm-bis] or on publish-subscribe messaging [I-D.ietf-core-coap-pubsub], and can be protected in different ways.

Therefore, this document delegates details on the communication and security approaches used in a group to separate application profiles. These are specialized instances of this document, targeting a particular group communication approach and defining how communications in the group are protected, as well as the specific keying material and configuration parameters provided to group members. In order to ensure consistency and aid the development of such application profiles, this document defines a number of related compliance requirements (see Appendix A).

If the application requires backward and forward security, new keying material is generated and distributed to the group upon membership changes (rekeying). A group rekeying scheme performs the actual distribution of the new keying material, by rekeying the current group members when a new Client joins the group, and the remaining group members when a Client leaves the group. This can rely on different approaches, including efficient group rekeying schemes such as [RFC2093], [RFC2094] and [RFC2627].

Consistently with what is recommended in the ACE framework, this document uses CBOR [RFC8949] for data encoding. However, using JSON [RFC8259] instead of CBOR is possible, by relying on the conversion method specified in Sections 6.1 and 6.2 of [RFC8949].

1.1. Terminology

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

Readers are expected to be familiar with:

- * The terms and concepts described in the ACE framework [I-D.ietf-ace-oauth-authz] and in the Authorization Information Format (AIF) [I-D.ietf-ace-aif] to express authorization information. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).
- * The terms and concepts described in CoAP [RFC7252]. Unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".
- * The terms and concepts described in CBOR [RFC8949] and COSE [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-algs] [I-D.ietf-cose-countersign].

A principal interested to participate in group communication as well as already participating as a group member is interchangeably denoted as "Client" or "node".

Furthermore, this document uses "names" or "identifiers" for groups and nodes. Their different meanings are summarized below.

- * **Group:** a set of nodes that share common keying material and security parameters used to protect their communications with one another. That is, the term refers to a "security group".

This is not to be confused with an "application group", which has relevance at the application level and whose members share a common pool of resources or content. Examples of application groups are the set of all nodes deployed in a same physical room, or the set of nodes registered to a pub-sub topic.

The same security group might be associated to multiple application groups. Also, the same application group can be associated to multiple security groups. Further details and considerations on the mapping between the two types of group are out of the scope of this document.

- * **Key Distribution Center (KDC):** the entity responsible for managing one or multiple groups, with particular reference to the group membership and the keying material to use for protecting group communications.
- * **Group name:** the invariant once established identifier of a group. It is used in the interactions between Client, AS and RS to identify a group. A group name is always unique among the group names of the existing groups under the same KDC.
- * **GROUPNAME:** the invariant once established text string used in URIs. GROUPNAME uniquely maps to the group name of a group, although they do not necessarily coincide.
- * **Group identifier:** the identifier of the group keying material used in a group. Unlike group name and GROUPNAME, this identifier changes over time, when the group keying material is updated.
- * **Node name:** the invariant once established identifier of a node. It is used in the interactions between Client and RS and to identify a member of a group. Within the same group, a node name is always unique among the node names of all the current members of that group.
- * **NODENAME:** the invariant once established text string used in URIs to identify a member a group. Its value coincides with the node name of the associated group member.

This document additionally uses the following terminology:

- * Transport profile, to indicate a profile of ACE as per Section 5.8.4.3 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. Transport 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.

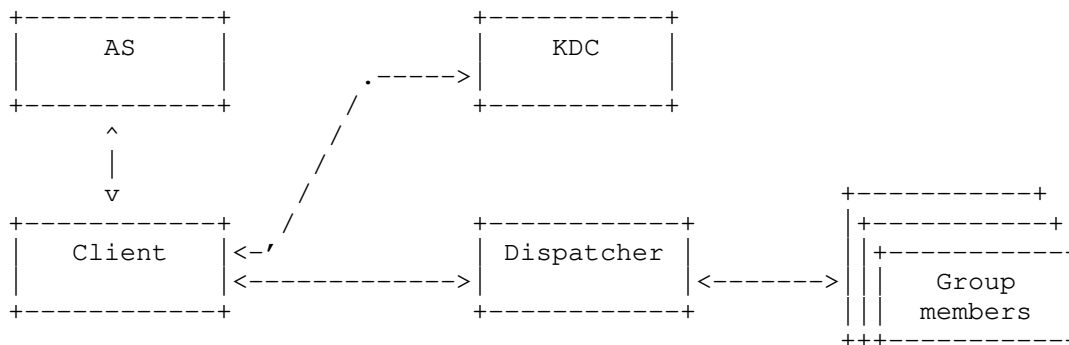


Figure 1: Key Distribution Participants

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

- * Client (C): node that wants to join a group and take part in group communication with other group members. Within the group, the Client can have different roles.
- * Authorization Server (AS): as per the AS defined 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, when sending a message intended to multiple group members. That is, the Dispatcher distributes such a one-to-many message to the group members as intended recipients. A single-recipient message intended to only one group member may be delivered by alternative means, with no assistance from the Dispatcher.

Examples of a Dispatcher are: the Broker in a pub-sub setting; a relay 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 Client 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 retrieve group keying material (Section 4.8.1.1 and Section 4.8.2.1).
- * Allowing a group member to retrieve public keys of other group members (Section 4.4.1.1) and to provide an updated public key (Section 4.9.1.1).
- * Allowing a group member to leave the group (Section 5).
- * Evicting a group member from the group (Section 5).
- * Renewing and re-distributing the group keying material (rekeying) upon a membership change in the group (Section 4.8.3.1 and Section 5).

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

1. The joining node requests an access token from the AS, in order to access one or more group-membership resources at the KDC and hence join the associated groups.

This exchange between Client and AS MUST be secured, as specified by the transport profile of ACE used between Client and KDC. Based on the response from the AS, the joining node will establish or continue using a secure communication association with the KDC.

2. The joining node transfers authentication and authorization information to the KDC, by transferring the obtained access token. This is typically achieved by including the access token in a request sent to the /authz-info endpoint at the KDC.

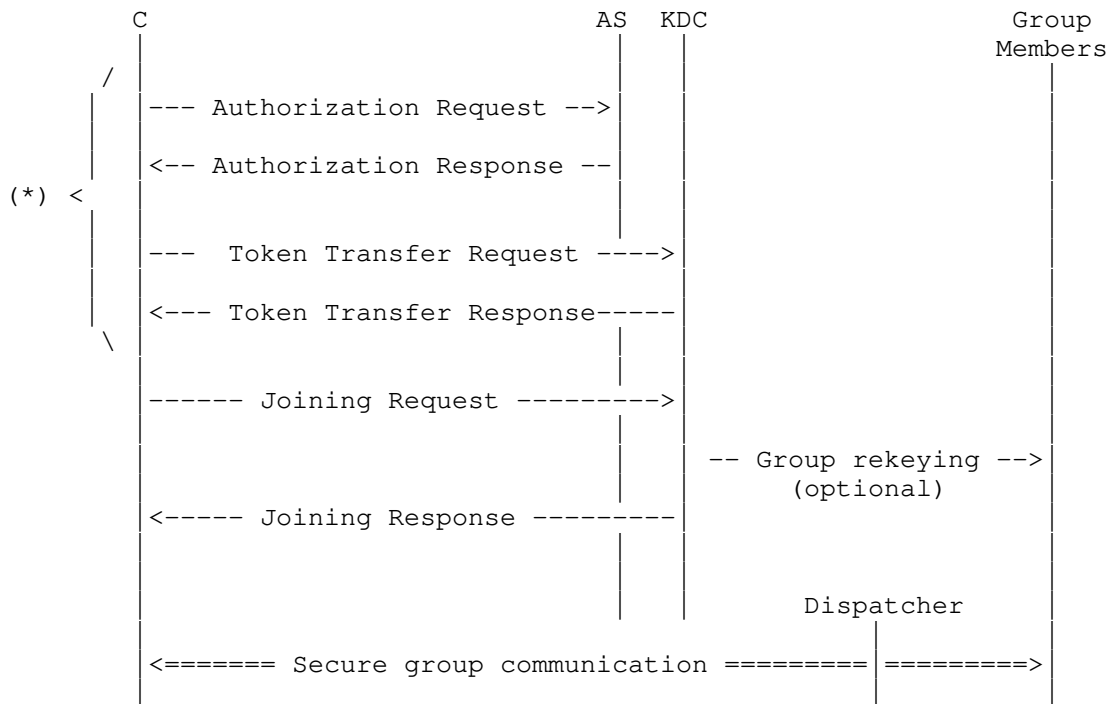
Once this exchange is completed, the joining node MUST have a secure communication association established with the KDC, before joining a group under that KDC.

This exchange and the following secure communications between the Client and the KDC MUST occur in accordance with the transport profile of ACE used between Client and KDC, such as 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 sending a request to the related group-membership resource at the KDC. Based on the application requirements and policies, the KDC may perform a group rekeying, by generating new group keying material and distributing it to the current group members through the rekeying scheme used in the group.

At the end of the joining process, the joining node has received from the KDC the parameters and group keying material to securely communicate with the other group members. Also, 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.



(*) Defined in the ACE framework

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 the AS for the 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, according to what is defined by the used transport profile of ACE. The Content-Format used in the message also depends on the used transport profile of ACE. For example, it 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.

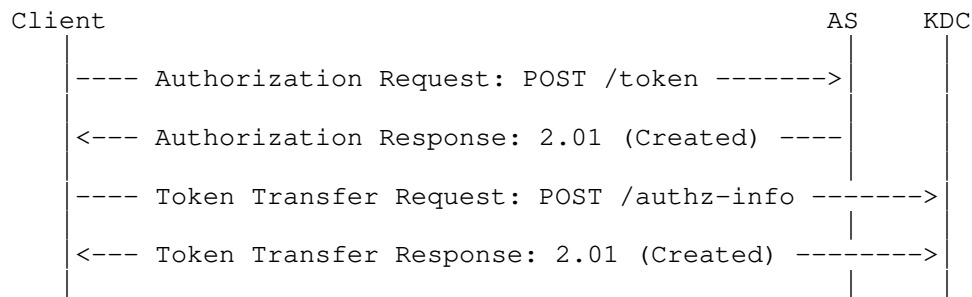


Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

The Authorization Request sent from the Client to the AS is defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz] and MAY contain the following parameters, which, if included, MUST have format and value as specified below.

- * 'scope', specifying the name of the groups that the Client requests to access, and optionally the roles that the Client requests to have in those groups.

This parameter is encoded as a CBOR byte string, which wraps a CBOR array of one or more scope entries. All the scope entries are specified according to a same format, i.e. either the AIF format or the textual format defined below.

- If the AIF format is used, each scope entry is encoded as specified in [I-D.ietf-ace-aif]. The object identifier "Toid" corresponds to the group name and MUST be encoded as a CBOR text string. The permission set "Tperm" indicates the roles that the Client wishes to take in the group.

The AIF format is the default format for application profiles of this specification, and is preferable for those that aim to a compact encoding of scope. This is desirable especially for application profiles defining several roles, with the Client possibly requesting for multiple roles combined.

Figure 4 shows an example in CDDL notation [RFC8610] where scope uses the AIF format.

- If the textual format is used, each scope entry is a CBOR array formatted as follows.
 - o As first element, the group name, encoded as a CBOR text string.
 - o 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).

Figure 5 shows an example in CDDL notation where scope uses the textual format, with group name and role identifiers encoded as CBOR text strings.

It is REQUIRED of application profiles of this specification to specify the exact format and encoding of scope (REQ1). This includes defining the set of possible roles and their identifiers, as well as the corresponding encoding to use in the scope entries according to the used scope format.

If the application profile uses the AIF format, it is also REQUIRED to register its specific instance of "Toid" and "Tperm", as well as the corresponding Media Type and Content-Format, as per the guidelines in [I-D.ietf-ace-aif] (REQ2).

If the application profile uses the textual format, it MAY additionally specify CBOR values to use for abbreviating the role identifiers (OPT1).

* 'audience', with an identifier of the 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 = &(amp;
    Requester: 1,
    Responder: 2,
    Monitor: 3,
    Verifier: 4
)

scope_entry = AIF_Generic<gname, permissions>

scope = << [ + scope_entry ] >>

```

Figure 4: Example of scope using the AIF format

```

gname = tstr

role = tstr

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

scope = << [ + scope_entry ] >>

```

Figure 5: Example of scope using the textual format, with the group name and role identifiers encoded as text strings

3.2. Authorization Response

The AS processes the Authorization Request as defined in Section 5.8.2 of [I-D.ietf-ace-oauth-authz], especially verifying that the Client is authorized to access the specified groups with the requested roles, or possibly a subset of those.

In case of successful verification, the Authorization Response sent from the AS to the Client is also defined in Section 5.8.2 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 8.14 of [I-D.ietf-ace-oauth-authz] for CWT).

This claim specifies the same access control information as in the 'scope' parameter of the Authorization Response, if the parameter is present in the message, or as in the 'scope' parameter of the Authorization Request otherwise.

By default, this claim has the same encoding as the 'scope' parameter in the Authorization Request, defined in Section 3.1.

Optionally, an alternative extended format of scope defined in Section 7 can be used. This format explicitly signals the semantics used to express the actual access control information, and according to which this has to be parsed. This enables a Resource Server to correctly process a received access token, also in case:

- The Resource Server implements a KDC that supports multiple application profiles of this specification, using different scope semantics; and/or
- The Resource Server implements further services beyond a KDC for group communication, using different scope semantics.

If the Authorization Server is aware that this applies to the Resource Server for which the access token is issued, the Authorization Server SHOULD use the extended format of scope defined in Section 7.

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 Transferring

The Client sends a Token Transfer Request to the KDC, i.e., a CoAP POST request including the access token and targeting the authz-info endpoint (see Section 5.10.1 of [I-D.ietf-ace-oauth-authz]).

Note that this request deviates from the one defined in [I-D.ietf-ace-oauth-authz], since it allows to ask the KDC for additional information concerning the public keys used in the group to ensure source authentication, as well as for possible additional group parameters.

The joining node MAY ask for this information from the KDC through the same Token Transfer Request. In this case, the message MUST have Content-Format set to application/ace+cbor defined in Section 8.16 of [I-D.ietf-ace-oauth-authz], and the message payload MUST be formatted as a CBOR map, which MUST include the access token. The CBOR map MAY additionally include the following parameter, which, if included, MUST have format and value as specified below.

- * 'sign_info' defined in Section 3.3.1, specifying the CBOR simple value 'null' (0xf6) to request information about the signature algorithm, signature algorithm parameters, signature key parameters and about the exact encoding of public keys used in the groups that the Client has been authorized to join.

Alternatively, such information may be pre-configured on the joining node, or may be retrieved by alternative means. For example, the joining node may have performed an early group discovery process and obtained the link to the associated group-membership resource at the KDC, together with attributes descriptive of the group configuration (see, e.g., [I-D.tiloca-core-oscore-discovery]).

After successful verification, the Client is authorized to receive the group keying material from the KDC and join the group. Hence, the KDC replies to the Client with a Token Transfer Response, i.e., a CoAP 2.01 (Created) response.

The Token Transfer Response MUST have Content-Format "application/ace+cbor", and its payload is a CBOR map. Note that this deviates from what is defined in the ACE framework, where the response from the authz-info endpoint is defined as conveying no payload (see Section 5.10.1 of [I-D.ietf-ace-oauth-authz]).

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.

Later, when joining the group (see Section 4.3.1.1), the Client uses the 'kdcchallenge' value and additional information to build a proof-of-possession (PoP) input. This is in turn used to compute a PoP evidence, which the Client also provides to the Group Manager in order to prove possession of its own private key (see the 'client_cred_verify' parameter in Section 4.3.1).

The KDC MUST store the 'kdcchallenge' value associated to the Client at least until it receives a Joining Request from it (see Section 4.3.1.1), to be able to verify the PoP evidence provided during the join process, and thus that the Client possesses its own private key.

The same 'kdcchallenge' value MAY be reused several times by the Client, to generate a new PoP evidence, e.g., in case the Client provides the Group Manager with a new public key while being a group member (see Section 4.9.1.1), or joins a different group where it intends to use a different public key. Therefore, it is RECOMMENDED that the KDC keeps storing the 'kdcchallenge' value after the first join is processed as well. If the KDC has already discarded the 'kdcchallenge' value, that will trigger an error response with a newly generated 'kdcchallenge' value that the Client can use to restart the join process, as specified in Section 4.3.1.1.

If 'sign_info' is included in the Token Transfer Request, the KDC SHOULD include the 'sign_info' parameter in the Token Transfer Response, as per the format defined in Section 3.3.1. Note that the field 'id' of each 'sign_info_entry' specifies the name, or array of group names, for which that 'sign_info_entry' applies to. As an exception, the KDC MAY omit the 'sign_info' parameter in the Token Transfer Response even if 'sign_info' is included in the Token Transfer Request, in case none of the groups that the Client is authorized to join uses signatures to achieve source authentication.

Note that the CBOR map specified as payload of the 2.01 (Created) response may include further parameters, e.g., according to the used transport profile of ACE. Application profiles of this specification MAY define additional parameters to use within this exchange (OPT2).

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 (OPT3).

If allowed by the used transport profile of ACE, the Client may provide the Access Token to the KDC by other means than the Token Transfer Request. An example is the DTLS transport profile of ACE, where the Client can provide the access token to the KDC during the secure session establishment (see Section 3.3.2 of [I-D.ietf-ace-dtls-authorize]).

3.3.1. 'sign_info' Parameter

The 'sign_info' parameter is an OPTIONAL parameter of the request and response messages exchanged between the Client and the authz-info endpoint at the RS (see Section 5.10.1. of [I-D.ietf-ace-oauth-authz]).

This parameter allows the Client and the RS to exchange information about a signature algorithm and about public keys to accordingly use for signature verification. Its exact semantics and content are application specific.

In this specification and in application profiles building on it, this parameter is used to exchange information about the signature algorithm and about public keys to be used with it, in the groups indicated by the transferred access token as per its 'scope' claim (see Section 3.2).

When used in the Token Transfer Request sent to the KDC (see Section 3.3), the 'sign_info' parameter specifies the CBOR simple value 'null' (0xf6). This is done to ask for information about the signature algorithm and about the public keys used in the groups that the Client has been authorized to join - or to have a more restricted interaction as per its granted roles (e.g., the Client is an external signature verifier).

When used in the following Token Transfer Response from the KDC (see Section 3.3), the 'sign_info' parameter 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 - or to have a more restricted interaction (see above). Each element contains information about signing parameters and about public keys for one or more groups, and is formatted as follows.

- * The first element 'id' is a group name or an array of group names, associated to groups for which the next four elements apply. In the following, 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 the CBOR simple value 'null' (0xf6), and indicates the signature algorithm used in the groups identified by the 'gname' values. 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 groups identified by the 'gname' values. 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 groups identified by the 'gname' values. 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 groups identified by the 'gname' values, or has value the CBOR simple value 'null' (0xf6) indicating that the KDC does not act as repository of public keys for group members. Its acceptable integer values are taken from the 'Label' column of the "COSE Header Parameters" registry [COSE.Header.Parameters]. It is REQUIRED of the application profiles to define specific values to use for this parameter, consistently with the acceptable formats of public keys (REQ6).

The CDDL notation [RFC8610] of the 'sign_info' parameter is given below.

```
sign_info = sign_info_req / sign_info_resp

sign_info_req = nil                                ; in the Token Transfer
                                                    ; Request to the KDC

sign_info_resp = [ + sign_info_entry ] ; in the Token Transfer
                                                    ; Response from the KDC

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

gname = tstr
```

This format is consistent with every signature algorithm currently defined in [I-D.ietf-cose-rfc8152bis-algs], i.e., with algorithms that have only the COSE key type as their COSE capability. Appendix B describes how the format of each 'sign_info_entry' can be generalized for possible future registered algorithms having a different set of COSE capabilities.

3.3.2. 'kdcchallenge' Parameter

The 'kdcchallenge' parameter is an OPTIONAL parameter of response message returned from the authz-info endpoint at the RS, as defined in Section 5.10.1 of [I-D.ietf-ace-oauth-authz]. This parameter contains a challenge generated by the RS and provided to the Client.

In this specification and in application profiles building on it, 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 4.3.1).

4. KDC Functionalities

This section describes the functionalities provided by the KDC, as related to the provisioning of the keying material as well as to the group membership management.

In particular, this section defines the interface available at the KDC; specifies the handlers of each resource provided by the KDC interface; and describes how Clients interact with those resources to join a group and to perform additional operations as group members.

As most important operation after transferring the access token to the KDC, the Client can perform a "Joining" exchange with the KDC, by specifying the group it requests to join (see Section 4.3.1.1). Then, the KDC verifies the access token and that the Client is authorized to join the specified group. If so, the KDC provides the Client with the keying material to securely communicate with the other members of the group.

Later on as a group member, the Client can also rely on the interface at the KDC to perform additional operations, consistently with the roles it has in the group.

4.1. Interface at the KDC

The KDC provides its interface by hosting the following resources. Note that the root url-path "ace-group" used hereafter is a default name; implementations are not required to use this name, and can define their own instead. The Interface Description (if=) Link Target Attribute value "ace.group" is registered in Section 11.5 and can be used to describe this interface.

If request messages sent to the KDC as well as success response messages from the KDC include a payload and specify a Content-Format, those messages MUST have Content-Format set to application/ace-groupcomm+cbor, defined in Section 11.2. CBOR labels for the message parameters are defined in Section 8.

- * /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, those applications will collide, and a mechanism will be needed to differentiate the endpoints.

A Client can access this resource in order to retrieve a set of group names, each corresponding to one of the specified group identifiers. This operation is described in Section 4.2.1.1.

- * /ace-group/GROUPNAME : one such sub-resource to /ace-group is hosted for each group with name GROUPNAME that the KDC manages, and contains the symmetric group keying material for that group.

A Client can access this resource in order to join the group with name GROUPNAME, or later as a group member to retrieve the current group keying material. These operations are described in Section 4.3.1.1 and Section 4.3.2.1, respectively.

If the value of the GROUPNAME URI path and the group name in the access token scope ('gname' in Section 3.2) are not required to coincide, the KDC MUST implement a mechanism to map the GROUPNAME value in the URI to the group name, in order to refer to the correct group (REQ7).

- * /ace-group/GROUPNAME/pub-key : this resource is invariant once established, and contains the public keys of all the members of the group with name GROUPNAME.

This resource is created only in case the KDC acts as repository of public keys for group members.

A Client can access this resource in order to retrieve the public keys of other group members, in addition to when joining the group. That is, the Client can retrieve the public keys of all the current group members, or a subset of them by specifying filter criteria. These operations are described in Section 4.4.2.1 and Section 4.4.1.1, respectively.

Clients may be authorized to access this resource even without being group members, e.g., if authorized to be external signature verifiers for the group.

- * ace-group/GROUPNAME/kdc-pub-key : this resource is invariant once established, and contains the public key of the KDC for the group with name GROUPNAME.

This resource is created only in case the KDC has an associated public key and this is required for the correct group operation. It is REQUIRED of application profiles to define whether the KDC has such an associated public key (REQ8).

A Client can interact with this resource in order to retrieve the current public key of the KDC, in addition to when joining the group.

Clients may be authorized to access this resource even without being group members, e.g., if authorized to be external signature verifiers for the group.

- * /ace-group/GROUPNAME/policies : this resource is invariant once established, and contains the group policies of the group with name GROUPNAME.

A Client can access this resource as a group member in order to retrieve the group policies. This operation is described in Section 4.6.1.1.

- * /ace-group/GROUPNAME/num : this resource is invariant once established, and contains the current version number for the symmetric group keying material of the group with name GROUPNAME.

A Client can access this resource as a group member in order to retrieve the version number of the keying material currently used in the group. This operation is described in Section 4.7.1.1.

- * /ace-group/GROUPNAME/nodes/NODENAME : one such sub-resource of /ace-group/GROUPNAME is hosted for each group member of the group with name GROUPNAME. Each of such resources is identified by the node name NODENAME of the associated group member, and contains the group keying material and the individual keying material for that group member.

A Client as a group member can access this resource in order to retrieve the current group keying material together with its the individual keying material; request new individual keying material to use in the group; and leave the group. These operations are described in Section 4.8.1.1, Section 4.8.2.1, and Section 4.8.3.1, respectively.

- * /ace-group/GROUPNAME/nodes/NODENAME/pub-key : this resource is invariant once established, and contains the individual public keying material for the node with name NODENAME, as group member of the group with name GROUPNAME.

A Client can access this resource in order to upload at the KDC a new public key to use in the group. This operation is described in Section 4.9.1.1.

This resource is not created if the group member does not have individual public keying material to use in the group, or if the KDC does not store the public keys of group members.

The KDC is expected to fully provide the interface defined above. It is otherwise REQUIRED of the application profiles of this specification to indicate which resources are not hosted, i.e., which parts of the interface defined in this section are not supported by the KDC (REQ9). Application profiles of this specification MAY extend the KDC interface, by defining additional resources and their handlers.

It is REQUIRED of the application profiles of this specification to register a Resource Type for the root url-path (REQ10). This 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.

It is REQUIRED of the application profiles of this specification to define what specific actions (e.g., CoAP methods) are allowed on each resource provided by the KDC interface, depending on whether the Client is a current group member; the roles that a Client is authorized to take as per the obtained access token (see Section 3.1); and the roles that the Client has as current group member (REQ11).

4.1.1. Operations Supported by Clients

It is expected that a Client minimally supports the following set of primary operations and corresponding interactions with the KDC.

- * FETCH request to ace-group/ , in order to retrieve group names associated to group identifiers.
- * POST and GET requests to ace-group/GROUPNAME/ , in order to join a group (POST) and later retrieve the current group key material as a group member (GET).
- * GET and FETCH requests to ace-group/GROUPNAME/pub-key , in order to retrieve the public keys of all the other group members (GET) or only some of them by filtering (FETCH). While retrieving public keys remains possible by using GET requests, retrieval by filtering allows to greatly limit the size of exchanged messages.
- * GET request to ace-group/GROUPNAME/num , in order to retrieve the current version of the group key material as a group member.
- * DELETE request to ace-group/GROUPNAME/nodes/NODENAME , in order to leave the group.

In addition, some Clients may rather not support the following set of secondary operations and corresponding interactions with the KDC. This can be specified, for instance, in compliance documents defining minimalistic Clients and their capabilities in specific deployments. In turn, these might also have to consider the used application profile of this specification.

- * GET request to `ace-group/GROUPNAME/kdc-pub-key` , in order to retrieve the current public key of the KDC, in addition to when joining the group. This is relevant only if the KDC has an associated public key and this is required for the correct group operation.
- * GET request to `ace-group/GROUPNAME/policies` , in order to retrieve the current group policies as a group member, in addition to when joining the group.
- * GET request to `ace-group/GROUPNAME/nodes/NODENAME`, in order to retrieve the current group keying material and individual keying material. The former can also be retrieved through a GET request to `ace-group/GROUPNAME/` (see above). The latter would not be possible to re-obtain as a group member.
- * PUT request to `ace-group/GROUPNAME/nodes/NODENAME` , in order to ask for new individual keying material. The Client would have to alternatively re-join the group through a POST request to `ace-group/GROUPNAME/` (see above). Furthermore, depending on its roles in the group or on the application profile of this specification, the Client might simply not be associated to any individual keying material.
- * POST request to `ace-group/GROUPNAME/nodes/NODENAME/pub-key` , in order to provide the KDC with a new public key. The Client would have to alternatively re-join the group through a POST request to `ace-group/GROUPNAME/` (see above). Furthermore, depending on its roles in the group, the Client might simply not have an associated public key to provide.

It is REQUIRED of application profiles of this specification to categorize possible newly defined operations for Clients into primary operations and secondary operations, and to provide accompanying considerations (REQ12).

4.1.2. Error Handling

Upon receiving a request from a Client, the KDC MUST check that it is storing a valid access token from that Client. If this is not the case, the KDC MUST reply with a 4.01 (Unauthorized) error response.

Unless the request targets the /ace-group resource, the KDC MUST check that it is storing a valid access token from that Client such that:

- * The scope specified in the access token includes a scope entry related to the group name GROUPNAME associated to targeted resource; and
- * The set of roles specified in that scope entry allows the Client to perform the requested operation on the targeted resource (REQ11).

In case the KDC stores a valid access token but the verifications above fail, the KDC MUST reply with a 4.03 (Forbidden) error response. This response MAY be an AS Request Creation Hints, as defined in Section 5.3 of [I-D.ietf-ace-oauth-authz], in which case the Content-Format MUST be set to application/ace+cbor.

If the request is not formatted correctly (e.g., required fields are not present or are not encoded as expected), the handler MUST reply with a 4.00 (Bad Request) error response.

If the request includes unknown or non-expected fields, the handler MUST silently ignore them and continue processing the request. Application profiles of this specification MAY define optional or mandatory payload formats for specific error cases (OPT4).

Some error responses from the KDC can have Content-Format set to application/ace-groupcomm+cbor. In such a case, the payload of the response MUST be a CBOR map, which includes the following fields.

- * 'error', with value a CBOR integer specifying the error occurred at the KDC. The value is taken from the "Value" column of the "ACE Groupcomm Errors" registry defined in Section 11.13 of this specification. This field MUST be present.
- * 'error_description', with value a CBOR text string specifying a human-readable diagnostic description of the error occurred at the KDC, written in English. The diagnostic text is intended for software engineers as well as for device and network operators, in order to aid debugging and provide context for possible intervention. The diagnostic message SHOULD be logged by the KDC. This field MAY be present, and it is unlikely relevant in an unattended setup where human intervention is not expected.

The 'error' and 'error_description' fields are defined as OPTIONAL to support for Clients (see Section 8). A Client supporting the 'error' parameter and able to understand the specified error may use that information to determine what actions to take next.

Section 9 of this specification defines an initial set of error identifiers, as possible values for the 'error' field. Application profiles of this specification inherit this initial set of error identifiers and MAY define additional value (OPT5).

4.2. /ace-group

This resource implements the FETCH handler.

4.2.1. 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, which 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 (REQ13). 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.

If all verifications succeed, the handler replies with a 2.05 (Content) response, whose payload is 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 to 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 verification on the group messages may be allowed to access this resource, if the application needs it.

4.2.1.1. Retrieve Group Names

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 4.2.1.

Figure 6 gives an overview of the exchanges described above, and Figure 7 shows an example.

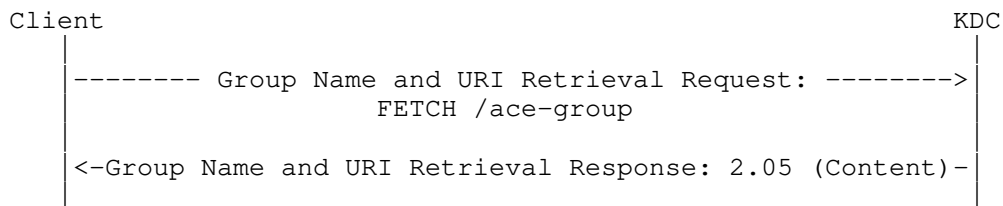


Figure 6: Message Flow of Group Name and URI Retrieval Request-Response

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": ["ace-group/g1", "ace-group/g2"] }
```

Figure 7: Example of Group Name and URI Retrieval Request-Response

4.3. /ace-group/GROUPNAME

This resource implements the POST and GET and handlers.

4.3.1. POST Handler

The POST handler processes the Joining Request sent by a Client to join a group, and returns a Joining Response as successful result of the joining process (see Section 4.3.1.1). At a high level, the POST handler adds the Client to the list of current group members, 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.

The handler expects a request with payload formatted as a CBOR map, which MAY contain the following fields, which, if included, MUST have format and value as specified below.

- * 'scope', with value the specific group that the Client is attempting to join, i.e., the group name, and the roles it wishes to have in the group. 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 current group members from the KDC. This parameter may be included in the Joining Request if the KDC stores the public keys of the group members, while it is not useful to include it if the Client obtains those public keys through alternative means, e.g., from the AS. Note that including this parameter might result in a following Joining Response of large size, which can be inconvenient for resource-constrained devices.

If the Client wishes to retrieve the public keys of all the current group members, the `'get_pub_keys'` parameter MUST encode the CBOR simple value `'null'` (0xf6). Otherwise, the `'get_pub_keys'` parameter MUST encode a non-empty CBOR array, containing the following three elements formatted as defined below.

- The first element, namely `'inclusion_flag'`, encodes the CBOR simple value True. That is, the Client indicates that it wishes to receive the public keys of all group members having their node identifier specified in the third element of the `'get_pub_keys'` array, namely `'id_filter'` (see below).
- The second element, namely `'role_filter'`, is a non-empty CBOR array. Each element of the array contains one role or a combination of roles for the group identified by GROUPNAME. That is, when the Joining Request includes a non-Null `'get_pub_keys'` parameter, the Client filters public keys based on node identifiers.

In particular, the Client indicates that it wishes to retrieve the public keys of all the group members having any of the single roles, or at least all of the roles indicated in any combination 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 third element, namely `'id_filter'`, is an empty CBOR array. That is, when the Joining Request includes a non-Null `'get_pub_keys'` parameter, the Client does not filter public keys based on node identifiers.

In fact, when first joining the group, the Client is not expected or capable to express a filter based on node identifiers of other group members. Instead, when already a group member and sending a Joining Request to re-join, the Client is not expected to include the 'get_pub_keys' parameter in the Joining Request altogether, since it can rather retrieve public keys associated to specific group identifiers as defined in Section 4.4.1.1.

The CDDL definition [RFC8610] of 'get_pub_keys' is given in Figure 8, 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, for this handler, 'inclusion_flag' is always set to true, the array of roles 'role_filter' is always non-empty, while the array of node identifiers 'id_filter' is always empty. However, this is not necessarily the case for other handlers using the 'get_pub_keys' parameter.

```
inclusion_flag = bool

role = tstr
comb_role = [ 2*role ]
role_filter = [ *(role / comb_role) ]

id = bstr
id_filter = [ *id ]

get_pub_keys = null / [ inclusion_flag, role_filter, id_filter]
```

Figure 8: CDDL definition of get_pub_keys, using as example node identifier encoded as bstr and role as tstr

- * 'client_cred', encoded as a CBOR byte string, with value the original binary representation of the Client's public key. This parameter 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. It is REQUIRED of the application profiles to define the specific formats that are acceptable to use for encoding public keys in the group (REQ6).
- * 'nonce', 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 proof-of-possession (PoP) evidence computed by the Client over the following PoP 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 Transfer 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 PoP input to compute `'client_cred_verify'` using CBOR encoding is given in Figure 9.

A possible type of PoP evidence is a signature, that the Client computes by using its own private key, whose corresponding public key is specified in the `'client_cred'` parameter. Application profiles of this specification MUST specify the exact approaches used to compute the PoP evidence to include in `'client_cred_verify'`, and MUST specify which of those approaches is used in which case (REQ14).

If the token was not provided to the KDC through a Token Transfer Request (e.g., it is used directly to validate TLS instead), it is REQUIRED of the specific application profile to define how the challenge `N_S` is generated (REQ15).

- * `'pub_keys_repos'`, which can be present if the format of the Client's public key in the `'client_cred'` parameter is a certificate. In such a case, this parameter has as value the URI of the certificate. This parameter is encoded as a CBOR text string. Alternative specific encodings of this parameter MAY be defined in applications of this specification (OPT6).

- * 'control_uri', with value a full URI, encoded as a CBOR text string. A default url-path is /ace-group/GROUPNAME/node, although implementations can use different ones instead. The URI MUST NOT have url-path ace-group/GROUPNAME.

If 'control_uri' is specified in the Joining Request, 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 one-to-one provisioning of new group keying material when performing a group rekeying (see Section 4.8.1.1), or to inform the Client of its removal from the group Section 5.

In particular, this resource is intended for communications concerning exclusively the group whose group name GROUPNAME is specified in the 'scope' parameter. If the KDC does not implement mechanisms using this resource for that group, it can ignore this parameter. Other additional functionalities of this resource MAY be defined in application profiles of this specifications (OPT7).

scope, N_S, and N_C expressed in CBOR diagnostic notation:

```
scope = h'8266667726F7570316673656E646572'
N_S   = h'018a278f7faab55a'
N_C   = h'25a8991cd700ac01'
```

scope, N_S, and N_C as CBOR encoded byte strings:

```
scope = 0x4f8266667726F7570316673656E646572
N_S   = 0x48018a278f7faab55a
N_C   = 0x4825a8991cd700ac01
```

PoP input:

```
0x4f 8266667726F7570316673656E646572
48 018a278f7faab55a 48 25a8991cd700ac01
```

Figure 9: Example of PoP input to compute 'client_cred_verify' using CBOR encoding

If the request does not include a 'scope' field, the KDC is expected to understand with what roles the Client is requesting to join the group. For example, as per the access token, the Client might have been granted access to the group with only one role. If the KDC cannot determine which exact scope should be considered for the Client, it MUST reply with a 4.00 (Bad Request) error response.

The handler considers the scope specified in the access token associated to the Client, and checks the scope entry related to the group with name GROUPNAME associated to the endpoint. In particular,

the handler checks whether the set of roles specified in that scope entry includes all the roles that the Client wishes to have in the group as per the Joining Request. If this is not the case, the KDC MUST reply with a 4.03 (Forbidden) error response.

If the KDC manages the group members' public keys, the handler checks if one is included in the 'client_cred' field. If so, the KDC retrieves the public key and performs the following actions.

- * If the access token was provided through a Token Transfer Request (see Section 3.3) but the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST reply with a 4.00 Bad Request error response, which MUST also have Content-Format application/ace-groupcomm+cbor. The payload of the error response is a CBOR map including a newly generated 'kdcchallenge' value. This is specified in the 'kdcchallenge' parameter.
- * The KDC checks the public key to be valid for the group identified by GROUPNAME. That is, it checks that the public key is encoded according to the format used in the group, is intended for the public key algorithm used in the group, and is aligned with the possible associated parameters used in the group.

If this verification fails, the handler MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 2 ("Public key incompatible with the group configuration").

- * The KDC verifies the PoP evidence contained in the 'client_cred_verify' field. Application profiles of this specification MUST specify the exact approaches used to verify the PoP evidence, and MUST specify which of those approaches is used in which case (REQ14).

If the PoP evidence does not pass verification, the handler MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 3 ("Invalid Proof-of-Possession evidence").

If no public key is included in the 'client_cred' field, the handler checks if a public key is already associated to the received access token and to the group identified by GROUPNAME (see also Section 4.3.1.1). Note that the same joining node may use different public keys in different groups, and all those public keys would be associate to the same access token.

If an eligible public key for the Client is neither present in the 'client_cred' field nor retrieved from the stored ones at the KDC, it is RECOMMENDED that the handler stops the processing and replies with a 4.00 (Bad Request) error response. Applications profiles MAY define alternatives (OPT8).

If, regardless the reason, the KDC replies with a 4.00 (Bad Request) error response, this response MAY have Content-Format set to application/ace-groupcomm+cbor and have a CBOR map as payload. For instance, the CBOR map can include a 'sign_info' parameter formatted as 'sign_info_res' defined in Section 3.3.1, with the 'pub_key_enc' element set to the CBOR simple value 'null' (0xf6) if the Client sent its own public key and the KDC is not set to store public keys of the group members.

If all the verifications above succeed, the KDC proceeds as follows.

First, only in case the Client is not already a group member, the handler performs the following actions:

- * The handler adds the Client to the list of current members of the group.
- * The handler assigns a name NODENAME to the Client, and creates a sub-resource to /ace-group/GROUPNAME at the KDC, i.e., "/ace-group/GROUPNAME/nodes/NODENAME".
- * The handler associates the node identifier NODENAME to the access token and the secure session for the Client.

Then, the handler performs the following actions.

- * If the KDC manages the group members' public keys:
 - The handler associates the retrieved Client's public key to the tuple composed of the node name NODENAME, the group name GROUPNAME and the received access token.
 - The handler adds the retrieved Client's public key to the stored list of public keys stored for the group identified by GROUPNAME. If such list already includes a public key for the Client, but a different public key is specified in the 'client_cred' field, then the handler MUST replace the old public key in the list with the one specified in the 'client_cred' field.

- * If the application requires backward security or if the used application profile prescribes so, the KDC MUST generate new group keying material and securely distribute it to the current group members (see Section 6).
- * The handler returns a successful Joining Response as defined below, containing the symmetric group keying material; the group policies; and the public keys of the current members of the group, if the KDC manages those and the Client requested them.

The Joining Response MUST have response code 2.01 (Created) if the Client has been added to the list of group members in this joining exchange (see above), or 2.04 (Changed) otherwise, i.e., if the Client is re-joining the group without having left it.

The Joining Response message MUST include the Location-Path CoAP option, specifying the URI path to the sub-resource associated to the Client, i.e. `"/ace-group/GROUPNAME/nodes/NODENAME"`.

The Joining Response message MUST have Content-Format `application/ace-groupcomm+cbor`. 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 Types" registry. Implementations MUST verify that the key type matches the application profile being used, if present, as registered in the "ACE Groupcomm Key Types" 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 (REQ16).

The exact format of the `'key'` value MUST be defined in applications of this specification (REQ17), as well as values of `'gkty'` accepted by the application (REQ18). Additionally, documents specifying the key format MUST register it in the "ACE Groupcomm Key Types" registry defined in Section 11.8, including its name, type and application profile to be used with.

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

Figure 10: 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 format and value as specified below.

- * `'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 Profiles" registry (REQ19).
- * `'pub_keys'`, MUST be present if `'get_pub_keys'` was present in the request, otherwise it MUST NOT be present. This parameter is a CBOR array specifying the public keys of the group members, i.e., of all of them or of the ones selected according to the `'get_pub_keys'` parameter in the request. In particular, each element of the array is a CBOR byte string, with value the original binary representation of a group member's public key. It is REQUIRED of the application profiles to define the specific formats of public keys that are acceptable to use in the group (REQ6).
- * `'peer_roles'`, MUST be present if `'pub_keys'` is also present, otherwise it MUST NOT be 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.

- * `'peer_identifiers'`, MUST be present if `'pub_keys'` is also present, otherwise it MUST NOT be 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 node identifier that the group member associated to the `i`-th public key in `'pub_keys'` has in the group. In particular, the `i`-th array element is encoded as a CBOR byte string, with value the node identifier of the group member.
- * `'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 Policies" registry. This specification defines the three elements "Sequence Number Synchronization Methods", "Key Update Check Interval" and "Expiration Delta", which are summarized in Figure 11. Application profiles that build on this document MUST specify the exact content format and default value of included map entries (REQ20).

Name	CBOR label	CBOR type	Description	Reference
Sequence Number Synchronization Method	TBD	tstr/int	Method for recipient group members to synchronize with sequence numbers of of sender group members. Its value is taken from the 'Value' column of the Sequence Number Synchronization Method registry	[[this document]]
Key Update Check Interval	TBD	int	Polling interval in seconds, for group members to check at the KDC if the latest group keying material is the one that they own	[[this document]]
Expiration Delta	TBD	uint	Number of seconds from 'exp' until the specified UTC date/time after which group members MUST stop using the group keying material they own to verify incoming messages	[[this document]]

Figure 11: ACE Groupcomm Policies

- * 'kdc_cred', encoded as a CBOR byte string, with value the original binary representation of the KDC's public key. This parameter is used if the KDC has an associated public key and this is required for the correct group operation. It is REQUIRED of application profiles to define whether the KDC has a public key and if this has to be provided through the 'kdc_cred' parameter (REQ8).

In such a case, the KDC's public key MUST have the same format used for the public keys of the group members. It is REQUIRED of the application profiles to define the specific formats that are acceptable to use for encoding public keys in the group (REQ6).

- * 'kdc_nonce', encoded as a CBOR byte string, and including a dedicated nonce N_KDC generated by the KDC. This parameter MUST be present if the 'kdc_cred' parameter is present.
- * 'kdc_cred_verify' parameter, encoded as a CBOR byte string. This parameter MUST be present if the 'kdc_cred' parameter is present.

This parameter contains a proof-of-possession (PoP) evidence computed by the KDC over the nonce N_KDC, which is specified in the 'kdc_nonce' parameter and taken as PoP input.

A possible type of PoP evidence is a signature, that the KDC computes by using its own private key, whose corresponding public key is specified in the 'kdc_cred' parameter. Application profiles of this specification MUST specify the exact approaches used by the KDC to compute the PoP evidence to include in 'kdc_cred_verify', and MUST specify which of those approaches is used in which case (REQ21).

- * 'rekeying_scheme', identifying the rekeying scheme that the KDC uses to provide new group keying material to the group members. This parameter is encoded as a CBOR integer, whose value is taken from the "Value" column of the "ACE Groupcomm Rekeying Schemes" registry defined in Section 11.14 of this specification.

Value	Name	Description	Reference
0	Point-to-Point	The KDC individually targets each node to rekey, using the pairwise secure communication association with that node	[this document]

Figure 12: ACE Groupcomm Rekeying Schemes

Application profiles of this specification MAY define a default group rekeying scheme, to refer to in case the 'rekeying_scheme' parameter is not included in the Joining Response (OPT9).

- * 'mgt_key_material', encoded as a CBOR byte string and containing the specific administrative keying material that the joining node requires in order to participate in the group rekeying process

performed by the KDC. This parameter MUST NOT be present if the 'rekeying_scheme' parameter is not present and the application profile does not specify a default group rekeying scheme to use in the group. Some simple rekeying scheme may not require specific administrative keying material to be provided, e.g., the basic "Point-to-Point" group rekeying scheme (see Section 6.1).

In more advanced group rekeying schemes, the administrative keying material can be composed of multiple keys organized, for instance, into a logical tree hierarchy, whose root key is the only administrative key shared by all the group members. In such a case, each group member is exclusively associated to one leaf key in the hierarchy, and owns only the administrative keys from the associated leaf key all the way up along the path to the root key. That is, different group members can be provided with a different subset of the overall administrative keying material.

It is expected from separate documents to define how the advanced group rekeying scheme possibly indicated in the 'rekeying_scheme' parameter is used by an application profile of this specification. This includes defining the format of the administrative keying material to specify in 'mgt_key_material', consistently with the group rekeying scheme and the application profile in question.

- * 'control_group_uri', with value a full URI, encoded as a CBOR text string. The URI MUST specify addressing information intended to reach all the members in the group. For example, this can be a multicast IP address, optionally together with a port number (which defaults to 5683 if omitted). The URI MUST include GROUPNAME in the url-path. A default url-path is /ace-group/GROUPNAME, although implementations can use different ones instead. The URI MUST NOT have url-path ace-group/GROUPNAME/node.

If 'control_group_uri' is included in the Joining Response, the Clients supporting this parameter act as CoAP servers, host a resource at this specific URI, and listen to the specified addressing information.

The KDC MAY use this URI to send one-to-many CoAP requests to the Client group members (acting as CoAP servers in this exchange), for example for one-to-many provisioning of new group keying material when performing a group rekeying (see Section 4.8.1.1), or to inform the Clients of their removal from the group
Section 5.

In particular, this resource is intended for communications concerning exclusively the group whose group name GROUPNAME is specified in the 'scope' parameter. If the KDC does not implement

mechanisms using this resource for that group, it can ignore this parameter. Other additional functionalities of this resource MAY be defined in application profiles of this specifications (OPT10).

If the Joining Response includes the 'kdc_cred_verify' parameter, the Client verifies the conveyed PoP evidence and considers the group joining unsuccessful in case of failed verification. Application profiles of this specification MUST specify the exact approaches used by the Client to verify the PoP evidence in 'kdc_cred_verify', and MUST specify which of those approaches is used in which case (REQ21).

Specific application profiles that build on this document MUST specify the communication protocol that members of the group use to communicate with each other (REQ22) and how exactly the keying material is used to protect the group communication (REQ23).

4.3.1.1. Join the Group

Figure 13 gives an overview of the Joining exchange between Client and KDC, when the Client first joins a group, while Figure 14 shows an example.

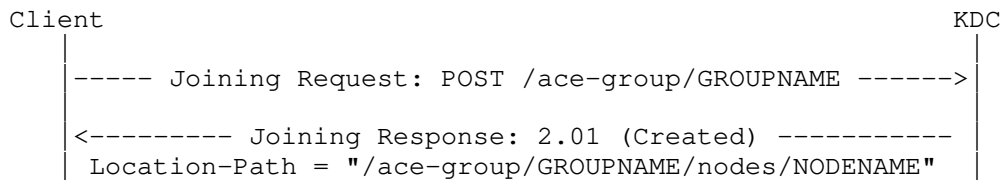


Figure 13: Message Flow of the Joining Exchange

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 POP_EVIDENCE being CBOR byte strings):
{ "scope": << [ "group1", ["sender", "receiver"] ] >> ,
  "get_pub_keys": [true, ["sender"], []], "client_cred": PUB_KEY,
  "cnonce": h'6df49c495409a9b5', "client_cred_verify": POP_EVIDENCE }
```

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"]],
  "peer_identifiers": [ ID1, ID2 ] }
```

Figure 14: Example of First Exchange for Group Joining

If not previously established, the Client and the KDC MUST first establish a pairwise secure communication channel (REQ24). 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 4.3.1. 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 individual 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 (PoP) evidence in the Joining Request (see the 'client_cred' and 'client_cred_verify' parameters in Section 4.3.1). The request is accepted only if both public key is provided and the PoP evidence is successfully verified.

If a node re-joins a group as authorized by the same access token and using the same public key, it can omit the public key and the PoP evidence, or just the PoP evidence, from the Joining Request. Then, the KDC will be able to retrieve the node's public key associated to the access token for that group. If the public key has been discarded, the KDC replies with 4.00 (Bad Request) error response, as specified in Section 4.3.1. If a node re-joins a group but wants to update its own public key, it needs to include both its public key and the PoP evidence in the Joining Request like when it joined the group for the first time.

4.3.2. GET Handler

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

The handler expects a GET request.

In addition to what is defined in Section 4.1.2, the handler verifies that the Client is a current member of the group. If the verification fails, the KDC MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").

If all verifications succeed, the handler replies with a 2.05 (Content) response 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 4.3.1.

Each of the following parameters specified in Section 4.3.1 MUST also be included in the payload of the response, if they are included in the payload of the Joining Responses sent for the group: 'rekeying_scheme', 'mgt_key_material'.

The payload MAY also include the parameters 'ace-groupcomm-profile' and 'exp' parameters specified in Section 4.3.1.

4.3.2.1. Retrieve Group Keying Material

A node in the group can contact the KDC to retrieve the current group keying material, by sending a CoAP GET request to the /ace-group/GROUPNAME endpoint at the KDC, where GROUPNAME is the group name.

Figure 15 gives an overview of the Joining exchange between Client and KDC, when the Client first joins a group, while Figure 16 shows an example.

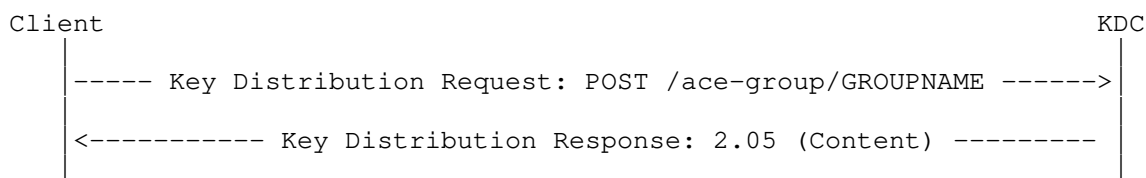


Figure 15: Message Flow of Key Distribution Request-Response

Request:

```

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

Response:

```

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation,
        with KEY being a CBOR byte strings):
{ "gkty": 13, "key": KEY, "num": 12 }
  
```

Figure 16: Example of Key Distribution Request-Response

4.4. /ace-group/GROUPNAME/pub-key

This resource implements the GET and FETCH handlers.

4.4.1. 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 field.

* 'get_pub_keys', whose value is encoded as in Section 4.3.1 with the following modifications.

- The arrays 'role_filter' and 'id_filter' MUST NOT both be empty, i.e., in CBOR diagnostic notation: [bool, [], []]. If the 'get_pub_keys' parameter has such a format, the request MUST be considered malformed, and the KDC MUST reply with a 4.00 (Bad Request) error response.

Note that a group member can retrieve the public keys of all the current group members by sending a GET request to the same KDC resource instead (see Section 4.4.2.1).

- The element 'inclusion_flag' encodes the CBOR simple value True if the third element 'id_filter' specifies an empty CBOR array, or if the Client wishes to receive the public keys of the nodes having their node identifier specified in 'id_filter' (i.e., selection by inclusive filtering). Instead, this element encodes the CBOR simple value False if the Client wishes to receive the public keys of the nodes not having the node identifiers specified in the third element 'id_filter' (i.e., selection by exclusive filtering).
- The array 'role_filter' can be empty, if the Client does not wish to filter the requested public keys based on the roles of the group members.
- The array 'id_filter' 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 the nodes having or not having these node identifiers, in case the 'inclusion_flag' element encodes the CBOR simple value True or False, respectively. The array 'id_filter' may be empty, if the Client does not wish to filter the requested public keys based on the node identifiers of the group members.

Note that, in case the 'role_filter' array and the 'id_filter' array are both non-empty:

- * If the 'inclusion_flag' encodes the CBOR simple value True, the handler returns the public keys of group members whose roles match with 'role_filter' and/or having their node identifier specified in 'id_filter'.
- * If the 'inclusion_flag' encodes the CBOR simple value False, the handler returns the public keys of group members whose roles match with 'role_filter' and, at the same time, not having their node identifier specified in 'id_filter'.

The specific format of public keys as well as identifiers, roles and combination of roles of group members MUST be specified by It is REQUIRED of application profiles of this specification (REQ1, REQ6, REQ25).

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.

If all verifications succeed, the handler returns a 2.05 (Content) message response with payload formatted as a CBOR map, containing only the following parameters from Section 4.3.1.

- * 'num', which encodes the version number of the current group keying material.
- * 'pub_keys', which encodes the list of public keys of the selected group members.
- * 'peer_roles', which encodes the role (or CBOR array of roles) that each of the selected group members has in the group.
- * 'peer_identifiers', which encodes the node identifier that each of the selected group members has in the group.

The specific format of public keys as well as of node identifiers of group members is specified by the application profile (REQ6, REQ25).

If the KDC does not store any public key associated to 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 'id_filter' element of 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 (REQ26).

- * 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.

If the KDC adopts this policy, the application profile MUST also specify the amount of time during which the KDC retains the public key of a former group member after its leaving, possibly on a per-member basis.

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.

4.4.1.1. Retrieve a Subset of Public Keys in the Group

In case the KDC maintains the public keys of group members, a node in the group can contact the KDC to request the public keys, roles and node identifiers of a specified subset of group members, by sending a CoAP 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 4.4.1.

Figure 17 gives an overview of the exchange mentioned above, while Figure 18 shows an example of such an exchange.

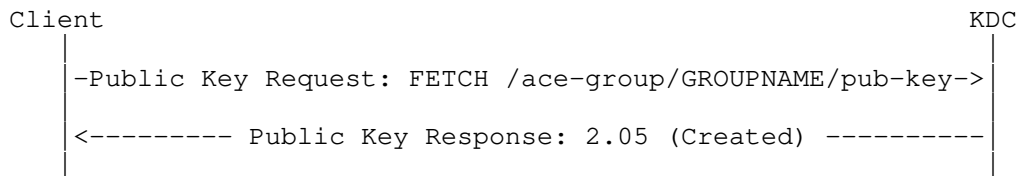


Figure 17: Message Flow of Public Key Exchange to Request the Public Keys of Specific Group Members

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": [true, [], [ ID3 ]] }
```

Response:

```
Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
  { "pub_keys": [ PUB_KEY3 ],
    "peer_roles": [ "receiver" ],
    "peer_identifiers": [ ID3 ] }
```

Figure 18: Example of Public Key Exchange to Request the Public Keys of Specific Group Members

4.4.2. GET Handler

The handler expects a GET request.

If all verifications succeed, the KDC replies with a 2.05 (Content) response as in the FETCH handler in Section 4.4.1, but specifying in the payload the public keys of all the group members, together with their roles and node identifiers.

4.4.2.1. Retrieve All Public Keys in the Group

In case the KDC maintains the public keys of group members, a group or an external signature verifier can contact the KDC to request the public keys, roles and node identifiers of all the current group members, by sending a CoAP GET request to the /ace-group/GROUPNAME/pub-key endpoint at the KDC, where GROUPNAME is the group name.

Figure 19 gives an overview of the message exchange, while Figure 20 shows an example of such an exchange.

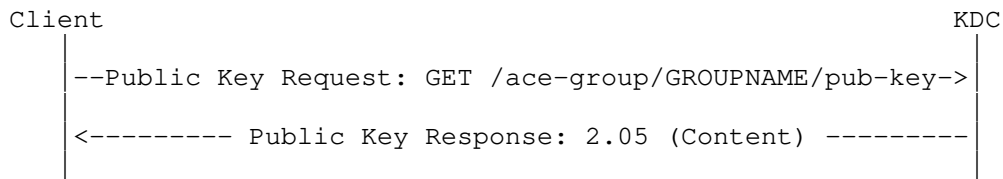


Figure 19: Message Flow of Public Key Exchange to Request the Public Keys of all the Group Members

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):
{ "num": 5,
  "pub_keys": [ PUB_KEY1, PUB_KEY2, PUB_KEY3 ],
  "peer_roles": ["sender", ["sender", "receiver"], "receiver"],
  "peer_identifiers": [ ID1, ID2, ID3 ] }
    
```

Figure 20: Example of Public Key Exchange to Request the Public Keys of all the Group Members

4.5. ace-group/GROUPNAME/kdc-pub-key

This resource implements a GET handler.

4.5.1. GET Handler

The handler expects a GET request.

If all verifications succeed, the handler returns a 2.05 (Content) message containing the KDC's public key together with a proof-of-possession (PoP) evidence. The response MUST have Content-Format set to application/ace-groupcomm+cbor. The payload of the response is a CBOR map, which includes the following fields.

- * The 'kdc_cred' parameter, specifying the KDC's public key. This parameter is encoded like the 'kdc_cred' parameter in the Joining Response (see Section 4.3.1).
- * The 'kdc_nonce' parameter, specifying a nonce generated by the KDC. This parameter is encoded like the 'kdc_nonce' parameter in the Joining Response (see Section 4.3.1).
- * The 'kdc_cred_verify' parameter, specifying a PoP evidence computed by the KDC. This parameter is encoded like the 'kdc_cred_verify' parameter in the Joining Response (see Section 4.3.1).

The PoP evidence is computed over the nonce specified in the 'kdc_nonce' parameter and taken as PoP input, by means of the same method used when preparing the Joining Response (see Section 4.3.1). Application profiles of this specification MUST specify the exact approaches used by the KDC to compute the PoP evidence to include in 'kdc_cred_verify', and MUST specify which of those approaches is used in which case (REQ21).

4.5.1.1. Retrieve the KDC's Public Key

In case the KDC has an associated public key as required for the correct group operation, a group member or an external signature verifier can contact the KDC to request the KDC's public key, by sending a CoAP GET request to the /ace-group/GROUPNAME/kdc-pub-key endpoint at the KDC, where GROUPNAME is the group name.

Upon receiving the 2.05 (Content) response, the Client retrieves the KDC's public key from the 'kdc_cred' parameter, and MUST verify the proof-of-possession (PoP) evidence specified in the 'kdc_cred_verify' parameter. In case of successful verification of the PoP evidence, the Client MUST store the obtained KDC's public key and replace the currently stored one.

The PoP evidence is verified by means of the same method used when processing the Joining Response (see Section 4.3.1). Application profiles of this specification MUST specify the exact approaches used by the Client to verify the PoP evidence in 'kdc_cred_verify', and MUST specify which of those approaches is used in which case (REQ21).

Figure 21 gives an overview of the exchange described above, while Figure 22 shows an example.

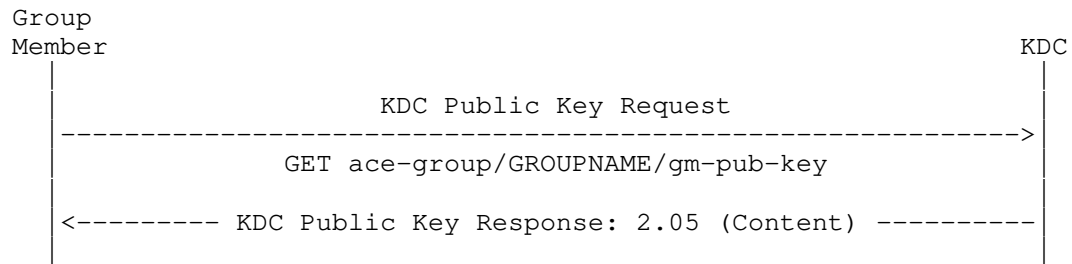


Figure 21: Message Flow of KDC Public Key Request-Response

Request:

```

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

Response:

```

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation, with PUB_KEY_KDC
        and POP_EVIDENCE being CBOR byte strings):
{
  "kdc_nonce": h'25a8991cd700ac01',
  "kdc_cred": PUB_KEY_KDC,
  "kdc_cred_verify": POP_EVIDENCE
}
    
```

Figure 22: Example of KDC Public Key Request-Response

4.6. /ace-group/GROUPNAME/policies

This resource implements the GET handler.

4.6.1. GET Handler

The handler expects a GET request.

In addition to what is defined in Section 4.1.2, the handler verifies that the Client is a current member of the group. If the verification fails, the KDC MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").

If all verifications succeed, the handler replies with a 2.05 (Content) response 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 4.3.1 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 (REQ20).

4.6.1.1. Retrieve the 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 4.6.1

Figure 23 gives an overview of the exchange described above, while Figure 24 shows an example.

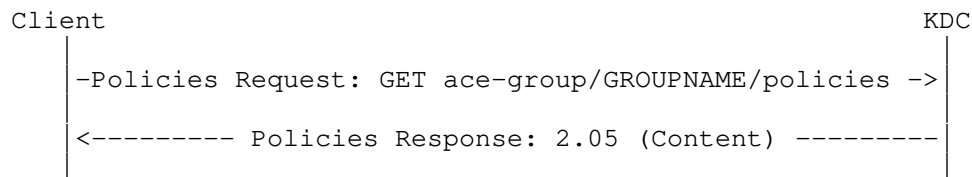


Figure 23: Message Flow of Policies Request-Response

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 24: Example of Policies Request-Response

4.7. /ace-group/GROUPNAME/num

This resource implements the GET handler.

4.7.1. GET Handler

The handler expects a GET request.

In addition to what is defined in Section 4.1.2, the handler verifies that the Client is a current member of the group. If the verification fails, the KDC MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").

If all verifications succeed, 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.

4.7.1.1. Retrieve the 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 `GROUENAME` is the group name, formatted as defined in Section 4.7.1. 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 25 gives an overview of the exchange described above, while Figure 26 shows an example.

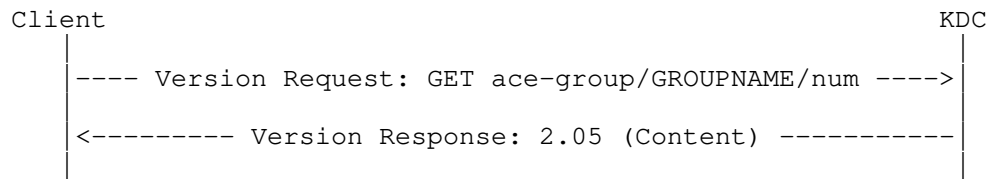


Figure 25: Message Flow of Version Request-Response

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: "application/ace-groupcomm+cbor"
Payload(in CBOR diagnostic notation):
  13
  
```

Figure 26: Example of Version Request-Response

4.8. `/ace-group/GROUPNAME/nodes/NODENAME`

This resource implements the GET, PUT and DELETE handlers.

In addition to what is defined in Section 4.1.2, each of the handlers performs the following two verifications.

- * The handler verifies that the Client is a current member of the group. If the verification fails, the KDC MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").
- * The handler verifies that the node name of the Client is equal to NODENAME used in the url-path. If the verification fails, the handler replies with a 4.03 (Forbidden) error response.

4.8.1. GET Handler

The handler expects a GET request.

If all verifications succeed, the handler replies with a 2.05 (Content) response 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 4.3.2. 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 (REQ27) and registered in Section 11.7.

Optionally, the KDC can make the sub-resource at ace-group/GROUPNAME/nodes/NODENAME also Observable [RFC7641] for the associated node. In case the KDC removes that node from the group without having been explicitly asked for it, this allows the KDC to send an unsolicited 4.04 (Not Found) response to the node as a notification of eviction from the group (see Section 5).

Note that the node could have been observing also the resource at ace-group/GROUPNAME, in order to be informed of changes in the keying material. In such a case, this method would result in largely overlapping notifications received for the resource at ace-group/GROUPNAME and the sub-resource at ace-group/GROUPNAME/nodes/NODENAME.

In order to mitigate this, a node that supports the No-Response option [RFC7967] can use it when starting the observation of the sub-resource at ace-group/GROUPNAME/nodes/NODENAME. In particular, the GET observation request can also include the No-Response option, with value set to 2 (Not interested in 2.xx responses).

4.8.1.1. Retrieve Group and Individual 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 4.8.1.

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-oscure-groupcomm]) to set up these policies for instructing Clients to retain incoming messages and for how long (OPT11). 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 27 gives an overview of the exchange described above, while Figure 28 shows an example.

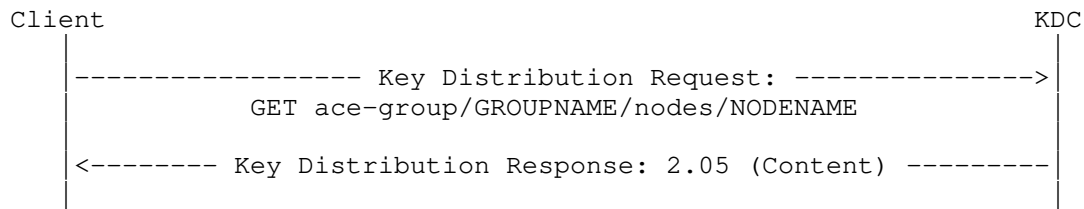


Figure 27: Message Flow of Key Distribution Request-Response

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 28: Example of Key Distribution Request-Response

4.8.2. PUT Handler

The PUT handler processes requests from a Client that asks for new individual keying material, as required to process messages exchanged in the group.

The handler expects a PUT request with empty payload.

In addition to what is defined in Section 4.1.2 and at the beginning of Section 4.8, the handler verifies that this operation is consistent with the set of roles that the Client has in the group (REQ11). If the verification fails, the KDC MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 1 ("Request inconsistent with the current roles").

If the KDC is currently not able to serve this request, i.e., to generate new individual keying material for the requesting Client, the KDC MUST reply with a 5.03 (Service Unavailable) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 4 ("No available node identifiers").

If all verifications succeed, the handler reply with a 2.05 (Content) response containing newly generated, individual keying material for the Client. 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 (REQ27) and registered in Section 11.7.

The typical successful outcome consists in replying with newly generated, individual keying material for the Client, as defined above. However, application profiles of this specification MAY also extend this handler in order to achieve different akin outcomes (OPT12), for instance:

- * Not providing the Client with newly generated, individual keying material, but rather rekeying the whole group, i.e., providing all the current group members with newly generated group keying material.
- * Both providing the Client with newly generated, individual keying material, as well as rekeying the whole group, i.e., providing all the current group members with newly generated group keying material.

In either case, the handler may specify the new group keying material as part of the 2.05 (Content) response.

Note that this handler is not intended to accommodate requests from a group member to trigger a group rekeying, whose scheduling and execution is an exclusive prerogative of the KDC.

4.8.2.1. Request to Change Individual Keying Material

A Client may ask the KDC for new, individual keying material. For instance, this can be due to the expiration of such individual keying material, or to the exhaustion of AEAD nonces, if an AEAD encryption algorithm is used for protecting communications in the group. An example of individual keying material can simply be an individual encryption key associated to the Client. Hence, the Client may ask for a new individual encryption key, or for new input material to derive it.

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 4.8.1.

Figure 29 gives an overview of the exchange described above, while Figure 30 shows an example.

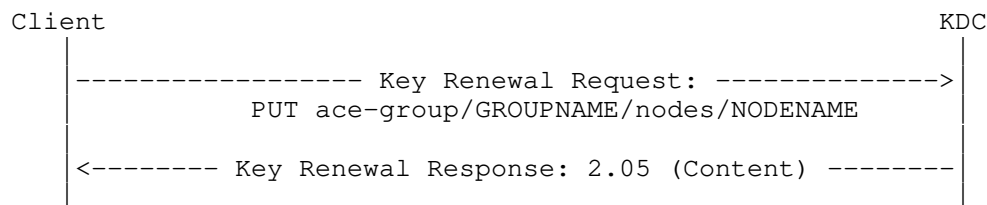


Figure 29: Message Flow of Key Renewal Request-Response

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 30: Example of Key Renewal Request-Response

Note the difference between the Key Renewal Request in this section and the Key Distribution Request in Section 4.8.1.1. The former asks the KDC for new individual keying material, while the latter asks the KDC for the current group keying material together with the current individual keying material.

As discussed in Section 4.8.2, application profiles of this specification may define alternative outcomes for the Key Renewal Request-Response exchange (OPT12), where the provisioning of new individual keying material is replaced by or combined with the execution of a whole group rekeying.

4.8.3. DELETE Handler

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

The handler expects a DELETE request with empty payload.

In addition to what is defined in Section 4.1.2, the handler verifies that the Client is a current member of the group. If the verification fails, the KDC MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").

If all verification succeeds, the handler performs the actions defined in Section 5 and replies with a 2.02 (Deleted) response with empty payload.

4.8.3.1. Leave the Group

A Client 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 4.8.3

Note that, after having left the group, the Client may wish to join it again. Then, as long as the Client is still authorized to join the group, i.e., the associated access token is still valid, the Client can request to re-join the group directly to the KDC (see Section 4.3.1.1), without having to retrieve a new access token from the AS.

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

This resource implements the POST handler.

4.9.1. 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 4.3.1, with the difference that it includes only the parameters 'client_cred', 'cnonce' and 'client_cred_verify'. In particular, the PoP evidence included in 'client_cred_verify' is computed in the same way considered in Section 4.3.1 and defined by the specific application profile (REQ14), with a newly generated N_C nonce and the previously received N_S. It is REQUIRED of the application profiles to define the specific formats of public keys that are acceptable to use in the group (REQ6).

In addition to what is defined in Section 4.1.2 and at the beginning of Section 4.8, the handler verifies that this operation is consistent with the set of roles that the node has in the group. If the verification fails, the KDC MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 1 ("Request inconsistent with the current roles").

If the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST reply with a 4.00 (Bad Request) error response, which MUST also have Content-Format application/ace-groupcomm+cbor. The payload of the error response is a CBOR map including a newly generated 'kdcchallenge' value. This is specified in the 'kdcchallenge' parameter. In such a case the KDC MUST store the newly generated value as the 'kdcchallenge' value associated to this Client, possibly replacing the currently stored value.

Otherwise, the handler checks that the public key specified in the 'client_cred' field is valid for the group identified by GROUPNAME. That is, the handler checks that the public key is encoded according to the format used in the group, is intended for the public key algorithm used in the group, and is aligned with the possible associated parameters used in the group. If that cannot be successfully verified, the handler MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 2 ("Public key incompatible with the group configuration").

Otherwise, the handler verifies the PoP evidence contained in the 'client_cred_verify' field of the request, by using the public key specified in the 'client_cred' field, as well as the same way considered in Section 4.3.1 and defined by the specific application profile (REQ14). If the PoP evidence does not pass verification, the handler MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 3 ("Invalid Proof-of-Possession evidence").

If all verifications succeed, the handler performs the following actions.

- * The handler associates the public key from the 'client_cred' field of the request to the node identifier NODENAME and to the access token associated to the node identified by NODENAME.
- * In the stored list of group members' public keys for the group identified by GROUPNAME, the handler replaces the public key of the node identified by NODENAME with the public key specified in the 'client_cred' field of the request.

Then, the handler replies with a 2.04 (Changed) response, which does not include a payload.

4.9.1.1. Uploading a New 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 4.9.1.

Figure Figure 31 gives an overview of the exchange described above, while Figure 32 shows an example.

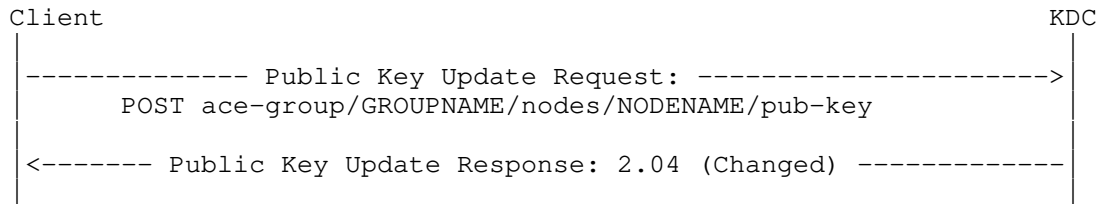


Figure 31: Message Flow of Public Key Update Request-Response

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 POP_EVIDENCE being CBOR byte strings):
{ "client_cred": PUB_KEY, "nonce": h'9ff7684414affcc8',
  "client_cred_verify": POP_EVIDENCE }

```

Response:

```

Header: Changed (Code=2.04)
Payload: -

```

Figure 32: Example of Public Key Update Request-Response

Additionally, after updating its own public key, a group member MAY send a number of requests including an identifier of the updated public key, to notify other group members that they have to retrieve it. How this is done depends on the group communication protocol used, and therefore is application profile specific (OPT13).

5. Removal of a Group Member

A Client identified by NODENAME may be removed from a group identified by GROUPNAME where it is a member, due to the following reasons.

1. The Client explicitly asks to leave the group, as defined in Section 4.8.3.1.
2. The node has been found compromised or is suspected so.
3. The Client's authorization to be a group member with the current roles is not valid anymore, i.e., the access token has expired or has been revoked. If the AS provides token introspection (see Section 5.9 of [I-D.ietf-ace-oauth-authz]), the KDC can optionally use it and check whether the Client is still authorized.

In either case, the KDC performs the following actions.

- * The KDC removes the Client from the list of current members or the group.
- * In case of forced eviction, i.e., for cases 2 and 3 above, the KDC deletes the public key of the removed Client, if it acts as repository of public keys for group members.
- * If the removed Client is registered as an observer of the group-membership resource at ace-group/GROUPNAME, the KDC removes the Client from the list of observers of that resource.
- * If the sub-resource nodes/NODENAME was created for the removed Client, the KDC deletes that sub-resource.

In case of forced eviction, i.e., for cases 2 and 3 above, the KDC MAY explicitly inform the removed Client, by means of the following methods.

- If the evicted Client implements the 'control_uri' resource specified in Section 4.3.1, the KDC sends a DELETE request, targeting the URI specified in the 'control_uri' parameter of the Joining Request (see Section 4.3.1).
- If the evicted Client is observing its associated sub-resource at ace-group/GROUPNAME/nodes/NODENAME (see Section 4.8.1), the KDC sends an unsolicited 4.04 (Not Found) error response, which does not include the Observe option and indicates that the observed resource has been deleted (see Section 3.2 of [RFC7641]).

The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 5 ("Group membership terminated").

- * If the application requires forward security or the used application profile requires so, the KDC MUST generate new group keying material and securely distribute it to all the current group members except the leaving node (see Section 6).

6. Group Rekeying Process

A group rekeying is started and driven by the KDC. The KDC is not intended to accommodate explicit requests from group members to trigger a group rekeying. That is, the scheduling and execution of a group rekeying is an exclusive prerogative of the KDC. Reasons that can trigger a group rekeying are a change in the group membership, the current group keying material approaching its expiration time, or a regularly scheduled update of the group keying material.

The KDC MUST increment the version number NUM of the current keying material, before distributing the newly generated keying material with version number NUM+1 to the group. Once completed the group rekeying, the KDC MUST delete the old keying material and SHOULD store the newly distributed keying material in persistent storage.

Distributing the new group keying material requires the KDC to send multiple rekeying messages to the group members. Depending on the rekeying scheme used in the group and the reason that has triggered the rekeying process, each rekeying message can be intended to one or multiple group members, hereafter referred to as target group members. The KDC MUST support at least the "Point-to-Point" group rekeying scheme in Section 6.1 and MAY support additional ones.

Each rekeying message MUST have Content-Format set to application/ace-groupcomm+cbor and its payload formatted as a CBOR map, which MUST include at least the information specified in the Key Distribution Response message (see Section 4.3.2), i.e., the parameters 'gkty', 'key' and 'num' defined in Section 4.3.1. The CBOR map MAY include the parameter 'exp', as well as the parameter 'mgt_key_material' specifying new administrative keying material for the target group members, if relevant for the used rekeying scheme.

A rekeying message may include additional information, depending on the rekeying scheme used in the group, the reason that has triggered the rekeying process and the specific target group members. In particular, if the group rekeying is performed due to one or multiple Clients that have joined the group and the KDC acts as repository of public keys of the group members, then a rekeying message MAY also include the public keys that those Clients use in the group, together with the roles and node identifier that the corresponding Client has in the group. It is RECOMMENDED to specify this information by means of the parameters 'pub_keys', 'peer_roles' and 'peer_identifiers', like done in the Joining Response message (see Section 4.3.1).

The complete format of a rekeying message, including the encoding and content of the 'mgt_key_material' parameter, has to be defined in separate specifications aimed at profiling the used rekeying scheme in the context of the used application profile of this specification. As a particular case, an application profile of this specification MAY define additional information to include in rekeying messages for the "Point-to-Point" group rekeying scheme in Section 6.1 (OPT14).

Consistently with the used group rekeying scheme, the actual delivery of rekeying messages can occur through different approaches, as discussed in the following.

6.1. Point-to-Point Group Rekeying

This approach consists in the KDC sending one individual rekeying message to each target group member. In particular, the rekeying message is protected by means of the security association between the KDC and the target group member in question, as per the used application profile of this specification and the used transport profile of ACE.

This is the approach taken by the basic "Point-to-Point" group rekeying scheme, that the KDC can explicitly signal in the Joining Response (see Section 4.3.1), through the 'rekeying_scheme' parameter specifying the value 0.

When taking this approach in the group identified by GROUPNAME, the KDC can practically deliver the rekeying messages to the target group members in different, co-existing ways.

- * The KDC SHOULD make the ace-group/GROUPNAME resource Observable [RFC7641]. Thus, upon performing a group rekeying, the KDC can distribute the new group keying material through individual notification responses sent to the target group members that are also observing that resource.

In case the KDC deletes the group, this also allows the KDC to send an unsolicited 4.04 (Not Found) response to each observer group member, as a notification of group termination. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4. The value of the 'error' field MUST be set to 6 ("Group deleted").

- * If a target group member specified a URI in the 'control_uri' parameter of the Joining Request upon joining the group (see Section 4.3.1), the KDC can provide that group member with the new group keying material by sending a unicast POST request to that URI.

A Client that does not plan to observe the ace-group/GROUPNAME resource at the KDC SHOULD provide a URI in the 'control_uri' parameter of the Joining Request upon joining the group.

If the KDC has to send a rekeying message to a target group member, but this did not include the 'control_uri' parameter in the Joining Request and is not a registered observer for the ace-group/GROUPNAME resource, then that target group member would not be able to participate to the group rekeying. Later on, after having repeatedly failed to successfully exchange secure messages in the group, that group member can retrieve the current group keying material from the KDC, by sending a GET request to ace-group/GROUPNAME or ace-group/GROUPNAME/nodes/NODENAME (see Section 4.3.2 and Section 4.8.1, respectively).

6.2. One-to-Many Group Rekeying

This section provides high-level recommendations on how the KDC can rekey a group by means of a more efficient and scalable group rekeying scheme, e.g., [RFC2093][RFC2094][RFC2627]. That is, each rekeying message might be, and likely is, intended to multiple target group members, and thus can be delivered to the whole group, although possible to decrypt only for the actual target group members.

This yields an overall lower number of rekeying messages, thus potentially reducing the overall time required to rekey the group. On the other hand, it requires the KDC to provide and use additional administrative keying material to protect the rekeying messages, and to additionally sign them to ensure source authentication (see Section 6.2.1). Typically, this pays off in large-scale groups, where the introduced performance overhead is less than what experienced by rekeying the group in a point-to-point fashion (see Section 6.1).

The exact set of rekeying messages to send, their content and format, the administrative keying material to use to protect them, as well as the set of target group members depend on the specific group rekeying scheme, and are typically affected by the reason that has triggered the group rekeying. Details about the data content and format of rekeying messages have to be defined by separate documents profiling the use of the group rekeying scheme, in the context of the used application profile of this specification.

When one of these group rekeying schemes is used, the KDC provides a number of related information to a Client joining the group in the Joining Response message (see Section 4.3.1). In particular, 'rekeying_scheme' identifies the rekeying scheme used in the group (if no default can be assumed); 'control_group_uri', if present, specifies a URI with a multicast address where the KDC will send the rekeying messages for that group; 'mgt_key_material' specifies a subset of the administrative keying material intended for that particular joining Client to have, as used to protect the rekeying messages sent to the group when intended also to that joining Client.

Rekeying messages can be protected at the application layer, by using COSE and the administrative keying material as prescribed by the specific group rekeying scheme (see Section 6.2.1). After that, the delivery of protected rekeying messages to the intended target group members can occur in different ways, such as the following ones.

- * Over multicast - In this case, the KDC simply sends a rekeying message as a CoAP request addressed to the multicast URI specified in the 'control_group_uri' parameter of the Joining Response (see Section 4.3.1).

If a particular rekeying message is intended to a single target group member, the KDC may alternatively protect the message using the security association with that group member, and deliver the message like when using the "Point-to-Point" group rekeying scheme (see Section 6.1).

- * Through a pub-sub communication model - In this case, the KDC acts as publisher and publishes each rekeying message to a specific "rekeying topic", which is associated to the group and is hosted at a broker server. Following their group joining, the group members subscribe to the rekeying topic at the broker, thus receiving the group rekeying messages as they are published by the KDC.

In order to make such message delivery more efficient, the rekeying topic associated to a group can be further organized into subtopics. For instance, the KDC can use a particular subtopic to

address a particular set of target group members during the rekeying process, as possibly aligned to a similar organization of the administrative keying material (e.g., a key hierarchy).

The setup of rekeying topics at the broker as well as the discovery of the topics at the broker for group members are application specific. A possible way is for the KDC to provide such information in the Joining Response message (see Section 4.3.1), by means of a new parameter analogous to 'control_group_uri' and specifying the URI(s) of the rekeying topic(s) that a group member has to subscribe to at the broker.

Regardless the specifically used delivery method, the group rekeying scheme can perform a possible roll-over of the administrative keying material through the same sent rekeying messages. Actually, such a roll-over occurs every time a group rekeying is performed upon the leaving of group members, which have to be excluded from future communications in the group.

From a high level point of view, each group member owns only a subset of the overall administrative keying material, obtained upon joining the group. Then, when a group rekeying occurs:

- * Each rekeying message is protected by using a (most convenient) key from the administrative keying material such that: i) the used key is not owned by any node leaving the group, i.e. the key is safe to use and does not have to be renewed; and ii) the used key is owned by all the target group members, that indeed have to be provided with new group keying material to protect communications in the group.
- * Each rekeying message includes not only the new group keying material intended to all the rekeyed group members, but also any new administrative keys that: i) are pertaining to and supposed to be owned by the target group members; and ii) had to be updated since leaving group members own the previous version.

Further details depend on the specific rekeying scheme used in the group.

6.2.1. Protection of Rekeying Messages

When using a group rekeying scheme relying on one-to-many rekeying messages, the actual data content of each rekeying message is prepared according to what the rekeying scheme prescribes.

Then, the KDC can protect the rekeying message as defined below. The used encryption algorithm which SHOULD be the same one used to protect communications in the group. The method defined below assumes that the following holds for the management keying material specified in the 'mgt_key_material' parameter of the Joining Response (see Section 4.3.1).

- * The included symmetric encryption keys are accompanied by a corresponding and unique key identifier assigned by the KDC.
- * A Base IV is also included, with the same size of the AEAD nonce considered by the encryption algorithm to use.

First, the KDC computes a COSE_Encrypt0 object as follows.

- * The encryption key to use is selected from the administrative keying material, as defined by the rekeying scheme used in the group.
- * The plaintext is the actual data content of the rekeying message.
- * The Additional Authenticated Data (AAD) is empty, unless otherwise specified by separate documents profiling the use of the group rekeying scheme.
- * Since the KDC is the only sender of rekeying messages, the AEAD nonce can be computed as follows, where NONCE_SIZE is the size in bytes of the AEAD nonce. Separate documents profiling the use of the group rekeying scheme may define alternative ways to compute the AEAD nonce.

The KDC considers the following values.

- COUNT, as a 1-byte unsigned integer associated to the used encryption key. Its value is set to 0 when starting to perform a new group rekeying instance, and is incremented after each use of the encryption key.
- NEW_NUM, as the version number of the new group keying material to distribute in this rekeying instance, left-padded with zeroes to exactly NONCE_SIZE - 1.

Then, the KDC computes a Partial IV as the byte string concatenation of COUNT and NEW_NUM, in this order. Finally, the AEAD nonce is computed as the XOR between the Base IV and the Partial IV.

- * The protected header of the COSE_Encrypt0 object MUST include the following parameters.
 - 'alg', specifying the used encryption algorithm.
 - 'kid', specifying the identifier of the encryption key from the administrative keying material used to protect this rekeying message.
- * The unprotected header of the COSE_Encrypt0 object MUST include the 'Partial IV' parameter, with value the Partial IV computed above.

In order to ensure source authentication, each rekeying message protected with the administrative keying material MUST be signed by the KDC. To this end, the KDC computes a countersignature of the COSE_Encrypt0 object, as described in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign]. In particular, the following applies when computing the countersignature.

- * The Countersign_structure contains the context text string "CounterSignature0".
- * The private key of the KDC is used as signing key.
- * The payload is the ciphertext of the COSE_Encrypt0 object.
- * The Additional Authenticated Data (AAD) is empty, unless otherwise specified by separate documents profiling the use of a group rekeying scheme.
- * The protected header of the signing object MUST include the parameter 'alg', specifying the used signature algorithm.

If source authentication of messages exchanged in the group is also ensured by means of signatures, then rekeying messages MUST be signed using the same signature algorithm and related parameters. Also, the KDC's public key used for signature verification MUST be provided in the Joining Response through the 'kdc_cred' parameter, together with the corresponding proof-of-possession (PoP) evidence in the 'kdc_cred_verify' parameter.

If source authentication of messages exchanged in the group is not ensured by means of signatures, then the KDC MUST provide its public key together with a corresponding PoP evidence as part of the management keying material specified in the 'mgt_key_material' parameter of the Joining Response (see Section 4.3.1). It is RECOMMENDED to specify this information by using the same format and

encoding used for the parameters 'kdc_cred', 'kdc_nonce' and 'kdc_cred_verify' in the Joining Response. It is up to separate documents profiling the use of the group rekeying scheme to specify such details.

After that, the KDC specifies the computed countersignature in the 'COSE_Countersignature0' header parameter of the COSE_Encrypt0 object.

Finally, the KDC specifies the COSE_Encrypt0 object as payload of a CoAP request, which is sent to the target group members as per the used message delivery method.

7. Extended Scope Format

This section defines an extended format of binary encoded scope, which additionally specifies the semantics used to express the same access control information from the corresponding original scope.

As also discussed in Section 3.2, this enables a Resource Server to unambiguously process a received access token, also in case the Resource Server runs multiple applications or application profiles that involve different scope semantics.

The extended format is intended only for the 'scope' claim of access tokens, for the cases where the claim takes as value a CBOR byte string. That is, the extended format does not apply to the 'scope' parameter included in ACE messages, i.e., the Authorization Request and Authorization Response exchanged between the Client and the Authorization Server (see Sections 5.8.1 and 5.8.2 of [I-D.ietf-ace-oauth-authz]), the AS Request Creation Hints message from the Resource Server (see Section 5.3 of [I-D.ietf-ace-oauth-authz]), and the Introspection Response from the Authorization Server (see Section 5.9.2 of [I-D.ietf-ace-oauth-authz]).

The value of the 'scope' claim following the extended format is composed as follows. Given the original scope using a semantics SEM and encoded as a CBOR byte string, the corresponding extended scope is encoded as a tagged CBOR byte string, wrapping a CBOR sequence [RFC8742] of two elements. In particular:

- * The first element of the sequence is a CBOR integer, and identifies the semantics SEM used for this scope. The value of this element has to be taken from the "Value" column of the "ACE Scope Semantics" registry defined in Section 11.12 of this specification.

When defining a new semantics for a binary scope, it is up to the applications and application profiles to define and register the corresponding integer identifier (REQ28).

- * The second element of the sequence is the original scope using the semantics SEM, encoded as a CBOR byte string.

Finally, the CBOR byte string wrapping the CBOR sequence is tagged, and identified by the CBOR tag TBD_TAG "ACE Extended Scope Format", defined in Section 11.6 of this specification.

The resulting tagged CBOR byte string is used as value of the 'scope' claim of the access token.

The usage of the extended scope format is not limited to application profiles of this specification or to applications based on group communication. Rather, it is generally applicable to any application and application profile where access control information in the access token is expressed as a binary encoded scope.

Figure 33 and Figure 34 build on the examples in Section 3.2, and show the corresponding extended scopes.

```

gname = tstr

permissions = uint . bits roles

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

scope_entry = AIF_Generic<gname, permissions>

scope = << [ + scope_entry ] >>

semantics = int

; This defines an array, the elements
; of which are to be used in a CBOR Sequence:
sequence = [semantics, scope]

extended_scope = #6.TBD_TAG(<< sequence >>)

```

Figure 33: Example CDLL definition of scope, using the default Authorization Information Format

```

gname = tstr

role = tstr

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

scope = << [ + scope_entry ] >>

semantics = int

; This defines an array, the elements
; of which are to be used in a CBOR Sequence:
sequence = [semantics, scope]

extended_scope = #6.TBD_TAG(<< sequence >>)

```

Figure 34: CDLL definition of scope, using as example group name encoded as tstr and role as tstr

8. ACE Groupcomm Parameters

This specification defines a number of parameters used during the second part of the message exchange, after the exchange of Token Transfer Request and Response. The table below summarizes them, and specifies the CBOR key to use instead of the full descriptive name.

Note that the media type application/ace-groupcomm+cbor MUST be used when these parameters are transported in the respective message fields.

Name	CBOR Key	CBOR Type	Reference
error	TBD	int	[this document]
error_description	TBD	tstr	[this document]
gid	TBD	array	[this document]
gname	TBD	array of tstr	[this document]
guri	TBD	array of tstr	[this document]
scope	TBD	bstr	[this document]
get_pub_keys	TBD	array / nil	[this document]

client_cred	TBD	bstr	[this document]
cnonce	TBD	bstr	[this document]
client_cred_verify	TBD	bstr	[this document]
pub_keys_repos	TBD	tstr	[this document]
control_uri	TBD	tstr	[this document]
gkty	TBD	int / tstr	[this document]
key	TBD	See the "ACE Groupcomm Key Types" registry	[this document]
num	TBD	int	[this document]
ace-groupcomm-profile	TBD	int	[this document]
exp	TBD	int	[this document]
pub_keys	TBD	array	[this document]
peer_roles	TBD	array	[this document]
peer_identifiers	TBD	array	[this document]
group_policies	TBD	map	[this document]
kdc_cred	TBD	bstr	[this document]
kdc_nonce	TBD	bstr	[this document]
kdc_cred_verify	TBD	bstr	[this document]
rekeying_scheme	TBD	int	[this document]
mgt_key_material	TBD	bstr	[this document]
control_group_uri	TBD	tstr	[this document]
sign_info	TBD	array	[this document]
kdcchallenge	TBD	bstr	[this document]

Figure 35: ACE Groupcomm Parameters

The KDC is expected to support and understand all the parameters above. Instead, a Client can support and understand only a subset of such parameters, depending on the roles it expects to take in the joined groups or on other conditions defined in application profiles of this specification.

In the following, the parameters are categorized according to the support expected by Clients. That is, a Client that supports a parameter is able to: i) use and specify it in a request message to the KDC; and ii) understand and process it if specified in a response message from the KDC. It is REQUIRED of application profiles of this specification to sort their newly defined parameters according to the same categorization (REQ29).

Note that the actual use of a parameter and its inclusion in a message depends on the specific exchange, the specific Client and group involved, as well as what is defined in the used application profile of this specification.

A Client MUST support the following parameters.

- * 'scope', 'gkty', 'key', 'num', 'exp', 'gid', 'gname', 'guri', 'pub_keys', 'peer_identifiers', 'ace_groupcomm_profile', 'control_uri', 'rekeying_scheme'.

A Client SHOULD support the following parameter.

- * 'get_pub_keys'. That is, not supporting this parameter would yield the inconvenient and undesirable behavior where: i) the Client does not ask for the other group members' public keys upon joining the group (see Section 4.3.1.1); and ii) later on as a group member, the Client only retrieves the public keys of all group members (see Section 4.4.2.1).

A Client MAY support the following optional parameters. Application profiles of this specification MAY define that Clients must or should support these parameters instead (OPT15).

- * 'error', 'error_description'.

The following conditional parameters are relevant only if specific conditions hold. It is REQUIRED of application profiles of this specification to define whether Clients must, should or may support these parameters, and under which circumstances (REQ30).

- * 'client_cred', 'cnonce', 'client_cred_verify'. These parameters are relevant for a Client that has a public key to use in a joined group.

- * `'kdcchallenge'`. This parameter is relevant for a Client that has a public key to use in a joined group and that provides the access token to the KDC through a Token Transfer Request (see Section 3.3).
- * `'pub_keys'repo'`. This parameter is relevant for a Client that has a public key to use in a joined group and that makes it available from a key repository different than the KDC.
- * `'group_policies'`. This parameter is relevant for a Client that is interested in the specific policies used in a group, but it does not know them or cannot become aware of them before joining that group.
- * `'peer_roles'`. This parameter is relevant for a Client that has to know about the roles of other group members, especially when retrieving and handling their corresponding public keys.
- * `'kdc_nonce'`, `'kdc_cred'`, `'kdc_cred_verify'`. These parameters are relevant for a Client that joins a group for which, as per the used application profile of this specification, the KDC has an associated public key and this is required for the correct group operation.
- * `'mgt_key_material'`. This parameter is relevant for a Client that supports an advanced rekeying scheme possibly used in the group, such as based on one-to-many rekeying messages sent over IP multicast.
- * `'control_group_uri'`. This parameter is relevant for a Client that supports the hosting of local resources each associated to a group (hence acting as CoAP server) and the reception of one-to-many requests sent to those resources by the KDC (e.g., over IP multicast), targeting multiple members of the corresponding group. Examples of related management operations that the KDC can perform by this means are the eviction of group members and the execution of a group rekeying process through an advanced rekeying scheme, such as based on one-to-many rekeying messages.

9. ACE Groupcomm Error Identifiers

This specification defines a number of values that the KDC can include as error identifiers, in the `'error'` field of an error response with Content-Format application/ace-groupcomm+cbor.

Value	Description
0	Operation permitted only to group members
1	Request inconsistent with the current roles
2	Public key incompatible with the group configuration
3	Invalid proof-of-possession evidence
4	No available node identifiers
5	Group membership terminated
6	Group deleted

Figure 36: ACE Groupcomm Error Identifiers

A Client supporting the 'error' parameter (see Section 4.1.2 and Section 8) and able to understand the specified error may use that information to determine what actions to take next. If it is included in the error response and supported by the Client, the 'error_description' parameter may provide additional context.

In particular, the following guidelines apply, and application profiles of this specification can define more detailed actions for the Client to take when learning that a specific error has occurred.

- * In case of error 0, the Client should stop sending the request in question to the KDC. Rather, the Client should first join the targeted group. If it has not happened already, this first requires the Client to obtain an appropriate access token authorizing access to the group and provide it to the KDC.
- * In case of error 1, the Client as a group member should re-join the group with all the roles needed to perform the operation in question. This might require the Client to first obtain a new access token and provide it to the KDC, if the current access token does not authorize to take those roles in the group. For operations admitted to a Client which is not a group member (e.g., an external signature verifier), the Client should first obtain a new access token authorizing to also have the missing roles.

- * In case of error 2, the Client has to obtain or self-generate a different asymmetric key pair, as aligned to the public key algorithms, parameters and encoding used in the targeted group. After that, the Client should provide its new consistent public key to the KDC.
- * In case of error 3, the Client should ensure to be computing its proof-of-possession evidence by correctly using the parameters and procedures defined in the used application profile of this specification. In an unattended setup, it might be not possible for a Client to autonomously diagnose the error and take an effective next action to address it.
- * In case of error 4, the Client should wait for a certain (pre-configured) amount of time, before trying re-sending its request to the KDC.
- * In case of error 5, the Client may try joining the group again. This might require the Client to first obtain a new access token and provide it to the KDC, e.g., if the current access token has expired.
- * In case of error 6, the Client should clean up its state regarding the group, just like if it has left the group with no intention to re-join it.

10. Security Considerations

Security considerations are inherited from the ACE framework [I-D.ietf-ace-oauth-authz], and from the specific transport profile of ACE used between the Clients and the KDC, e.g., [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

Furthermore, the following security considerations apply.

10.1. Secure Communication in the Group

When a group member receives a message from a certain sender for the first time since joining the group, it needs to have a mechanism in place to avoid replayed messages, e.g., Appendix B.2 of [RFC8613] or Appendix E of [I-D.ietf-core-oscore-groupcomm]. Such a mechanism aids the recipient group member also in case it has rebooted and lost the security state used to protect previous group communications with that sender.

By its nature, the KDC is invested with a large amount of trust, since it acts as generator and provider of the symmetric keying material used to protect communications in each of its groups. While

details depend on the specific communication and security protocols used in the group, the KDC is in the position to decrypt messages exchanged in the group as if it was also a group member, as long as those are protected through commonly shared group keying material.

A compromised KDC would thus put the attacker in the same position, which also means that:

- * The attacker can generate and control new group keying material, hence possibly rekeying the group and evicting certain group members as part of a broader attack.
- * The attacker can actively participate to communications in a group even without been authorized to join it, and can allow further unauthorized entities to do so.
- * The attacker can build erroneous associations between node identifiers and group members' public keys.

On the other hand, as long as the security protocol used in the group ensures source authentication of messages (e.g., by means of signatures), the KDC is not able to impersonate group members since it does not own their private keys.

Further security considerations are specific of the communication and security protocols used in the group, and thus have to be provided by those protocols and complemented by the application profiles of this specification using them.

10.2. Update of Group Keying Material

Due to different reasons, the KDC can generate new group keying material and provide it to the group members (rekeying) through the rekeying scheme used in the group, as discussed in Section 6.

In particular, the KDC must renew the group keying material latest upon its expiration. Before then, the KDC may also renew the group keying material on a regular or periodical fashion.

The KDC should renew the group keying material upon a group membership change. Since the minimum number of group members is one, the KDC should provide also a Client joining an empty group with new keying material never used before in that group. Similarly, the KDC should provide new group keying material also to a Client that remains the only member in the group after the leaving of other group members.

Note that the considerations in Section 10.1 about dealing with replayed messages still hold, even in case the KDC rekeys the group upon every single joining of a new group member. However, if the KDC has renewed the group keying material upon a group member's joining, and the time interval between the end of the rekeying process and that member's joining is sufficiently small, then that group member is also on the safe side, since it would not accept replayed messages protected with the old group keying material previous to its joining.

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 each and every group membership change, for instance if members' joining and leaving occur frequently and performing a group rekeying takes too long. Instead, the KDC might rekey the group after a minimum number of group members have joined or left within a given time interval, or after a maximum amount of time since the last group rekeying was completed, or yet during predictable network inactivity periods.

However, this would result in the KDC not constantly preserving backward and forward security in the group. That is:

- * Newly joining group members would be able to access the keying material used before their joining, and thus they could access past group communications if they have recorded old exchanged messages. This might still be acceptable for some applications and in situations where the new group members are freshly deployed through strictly controlled procedures.
- * The leaving group members would remain able to access upcoming group communications, as protected with the current keying material that has not been updated. This is typically undesirable, especially if the leaving group member is compromised or suspected to be, and it might have an impact or compromise the security properties of the protocols used in the group to protect messages exchanged among the group member.

The KDC should renew the group keying material in case it has rebooted, even in case it stores the whole group keying material in persistent storage. This assumes that the secure associations with the current group members as well as any administrative keying material required to rekey the group are also stored in persistent storage.

However, if the KDC relies on Observe notifications to distribute the new group keying material, the KDC would have lost all the current ongoing Observations with the group members after rebooting, and the

group members would continue using the old group keying material. Therefore, the KDC will rather rely on each group member asking for the new group keying material (see Section 4.3.2.1 and Section 4.8.1.1), or rather perform a group rekeying by actively sending rekeying messages to group members as discussed in Section 6.

The KDC needs to have a mechanism in place to detect DoS attacks from nodes repeatedly performing actions that might trigger a group rekeying. Such actions can include leaving and/or re-joining the group at high rates, or often asking the KDC for new individual keying material. Ultimately, the KDC can resort to removing these nodes from the group and (temporarily) preventing them from joining the group again.

The KDC also needs to have a congestion control mechanism in place, in order to avoid network congestion upon distributing new group keying material. For example, CoAP and Observe give guidance on such mechanisms, see Section 4.7 of [RFC7252] and Section 4.5.1 of [RFC7641].

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

10.2.1. Misalignment of Group 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 (see Section 6). 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 group keying material. However, the recipient receives the message after having received the new group keying material, hence not being able to correctly process it. A possible way to ameliorate this issue is to preserve the old, recent group keying material for a maximum amount of time defined by the application, during which it is used solely for processing incoming messages. By doing so, the recipient can still temporarily process received messages also by using the old, retained group keying material. Note that a former (compromised) group member can take advantage of this by sending messages protected with the old, retained group keying material. Therefore, a conservative application policy should not admit the storage of old group keying material. Eventually, the sender will have obtained the new group keying material too, and can possibly re-send the message protected with such keying material.

In the second case, the sender protects a message using the new group keying material, but the recipient receives that message before having received the new group keying material. Therefore, the recipient would not be able to correctly process the message and hence discards it. If the recipient receives the new group 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 the latest group keying material according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

10.3. Block-Wise Considerations

If the Block-Wise CoAP 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 group keying material to the updated one and continues the transfer. As long as both sides get the new group keying material, updating group 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 group keying material is changed, the group members might consume a larger amount of the network bandwidth by repeatedly resending the same blocks, which might be problematic.

11. IANA Considerations

This document has the following actions for IANA.

11.1. 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: N/A

Optional parameters: N/A

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

Security considerations: See Section 10 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].

Fragment identifier considerations: N/A

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

11.2. CoAP Content-Formats

IANA is asked to register the following entry to the "CoAP Content-Formats" registry within the "CoRE Parameters" registry group.

Media Type: application/ace-groupcomm+cbor

Encoding: -

ID: TBD

Reference: [this document]

11.3. OAuth Parameters

IANA is asked to register the following entries in the "OAuth Parameters" registry following the procedure specified in Section 11.2 of [RFC6749].

- * Parameter name: sign_info
- * Parameter usage location: client-rs request, rs-client response
- * Change Controller: IESG
- * Specification Document(s): [[This specification]]

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

11.4. OAuth Parameters CBOR Mappings

IANA is asked to register the following entries in the "OAuth Parameters CBOR Mappings" registry following the procedure specified in Section 8.10 of [I-D.ietf-ace-oauth-authz].

- * Name: sign_info
- * CBOR Key: TBD (range -256 to 255)
- * Value Type: Simple value null / array
- * Reference: [[This specification]]

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

11.5. Interface Description (if=) Link Target Attribute Values

IANA is asked to register the following entry in the "Interface Description (if=) Link Target Attribute Values" registry within the "CoRE Parameters" registry group.

- * Attribute Value: ace.group

- * Description: The 'ace group' interface is used to provision keying material and related information and policies to members of a group using the Ace framework.
- * Reference: [This Document]

11.6. CBOR Tags

IANA is asked to register the following entry in the "CBOR Tags" registry.

- * Tag : TBD_TAG
- * Data Item: byte string
- * Semantics: Extended ACE scope format, including the identifier of the used scope semantics.
- * Reference: [This Document]

11.7. ACE Groupcomm Parameters

This specification establishes the "ACE Groupcomm Parameters" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15.

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 8. The Reference column for all of these entries refers to sections of this document.

11.8. ACE Groupcomm Key Types

This specification establishes the "ACE Groupcomm Key Types" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15.

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 Profiles" 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 10. The specification column for all of these entries will be this document.

11.9. ACE Groupcomm Profiles

This specification establishes the "ACE Groupcomm Profiles" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to 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.

11.10. ACE Groupcomm Policies

This specification establishes the "ACE Groupcomm Policies" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to 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 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 11.

11.11. Sequence Number Synchronization Methods

This specification establishes the "Sequence Number Synchronization Methods" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to 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.

11.12. ACE Scope Semantics

This specification establishes the "ACE Scope Semantics" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to be supplied as well.

The columns of this registry are:

- * Value: The value to be used to identify this scope semantics. The value MUST be unique. The value can be a positive integer or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values 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.
- * Description: This field contains a brief description of the scope semantics.

- * Reference: This field contains a pointer to the public specification defining the scope semantics, if one exists.

11.13. ACE Groupcomm Errors

This specification establishes the "ACE Groupcomm Errors" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to be supplied as well.

The columns of this registry are:

- * Value: The value to be used to identify the error. The value MUST be unique. The value can be a positive integer or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values 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.
- * Description: This field contains a brief description of the error.
- * Reference: This field contains a pointer to the public specification defining the error, if one exists.

This registry has been initially populated by the values in Section 9. The Reference column for all of these entries refers to this document.

11.14. ACE Groupcomm Rekeying Schemes

This specification establishes the "ACE Groupcomm Rekeying Schemes" IANA registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.15. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, to be supplied as well.

The columns of this registry are:

- * Value: The value to be used to identify the group rekeying scheme. The value MUST be unique. The value can be a positive integer or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values

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.

- * Name: The name of the group rekeying scheme.
- * Description: This field contains a brief description of the group rekeying scheme.
- * Reference: This field contains a pointer to the public specification defining the group rekeying scheme, if one exists.

This registry has been initially populated by the value in Figure 12.

11.15. 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.

12. References

12.1. Normative References

- [COSE.Algorithms]
IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.
- [COSE.Header.Parameters]
IANA, "COSE Header Parameters",
<<https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>>.
- [I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", Work in Progress, Internet-Draft, draft-ietf-ace-aif-03, 24 June 2021,
<<https://www.ietf.org/archive/id/draft-ietf-ace-aif-03.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-46, 8 November 2021,
<<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-13, 25 October 2021,
<<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-13.txt>>.

- [I-D.ietf-cose-countersign]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.
- [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>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

- [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>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [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>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

12.2. 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-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.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-13, 23 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-mqtt-tls-profile-13.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-19, 6 May
2021, <[https://www.ietf.org/archive/id/draft-ietf-ace-
oscore-profile-19.txt](https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.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,
<[https://www.ietf.org/archive/id/draft-ietf-core-coap-
pubsub-09.txt](https://www.ietf.org/archive/id/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-
05, 25 October 2021, <[https://www.ietf.org/archive/id/
draft-ietf-core-groupcomm-bis-05.txt](https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-05.txt)>.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of
OSCORE Groups with the CoRE Resource Directory", Work in
Progress, Internet-Draft, draft-tiloca-core-oscore-
discovery-10, 25 October 2021,
<[https://www.ietf.org/archive/id/draft-tiloca-core-oscore-
discovery-10.txt](https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-10.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>>.
- [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.

A.1. Mandatory-to-Address Requirements

- * REQ1: Specify the format and encoding of 'scope'. This includes defining the set of possible roles and their identifiers, as well as the corresponding encoding to use in the scope entries according to the used scope format (see Section 3.1).
- * REQ2: If the AIF format of 'scope' is used, register its specific instance of "Toid" and "Tperm", as well as the corresponding Media Type and Content-Format, as per the guidelines in [I-D.ietf-ace-aif].
- * 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: Specify the acceptable formats for encoding public keys and, if used, the acceptable values for 'pub_key_enc' (see Section 3.3).
- * REQ7: If the value of the GROUPNAME URI path and the group name in the access token scope (gname in Section 3.2) are not required to coincide, specify the mechanism to map the GROUPNAME value in the URI to the group name (see Section 4.1).
- * REQ8: Define whether the KDC has a public key and if this has to be provided through the 'kdc_cred' parameter, see Section 4.3.1.
- * REQ9: Specify if any part of the KDC interface as defined in this document is not supported by the KDC (see Section 4.1).
- * REQ10: 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).
- * REQ11: Define what specific actions (e.g., CoAP methods) are allowed on each resource provided by the KDC interface, depending on whether the Client is a current group member; the roles that a Client is authorized to take as per the obtained access token (see Section 3.1); and the roles that the Client has as current group member.
- * REQ12: Categorize possible newly defined operations for Clients into primary operations expected to be minimally supported and secondary operations, and provide accompanying considerations (see Section 4.1.1).
- * REQ13: Specify the encoding of group identifier (see Section 4.2.1).
- * REQ14: Specify the approaches used to compute and verify the PoP evidence to include in 'client_cred_verify', and which of those approaches is used in which case (see Section 4.3.1).
- * REQ15: Specify how the nonce N_S is generated, if the token is not provided to the KDC through the Token Transfer Request to the authz-info endpoint (e.g., if it is used directly to validate TLS instead).

- * REQ16 Define the initial value of the 'num' parameter (see Section 4.3.1).
- * REQ17: Specify the format of the 'key' parameter (see Section 4.3.1).
- * REQ18: Specify the acceptable values of the 'gkty' parameter (see Section 4.3.1).
- * REQ19: Specify and register the application profile identifier (see Section 4.3.1).
- * REQ20: If used, specify the format and content of 'group_policies' and its entries. Specify the policies default values (see Section 4.3.1).
- * REQ21: Specify the approaches used to compute and verify the PoP evidence to include in 'kdc_cred_verify', and which of those approaches is used in which case (see Section 4.3.1).
- * REQ22: Specify the communication protocol the members of the group must use (e.g., multicast CoAP).
- * REQ23: 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.
- * REQ24: Specify how the communication is secured between Client and KDC. Optionally, specify transport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and KDC (see Section 4.3.1.1).
- * REQ25: Specify the format of the identifiers of group members (see Section 4.3.1).
- * REQ26: Specify policies at the KDC to handle ids that are not included in 'get_pub_keys' (see Section 4.4.1).
- * REQ27: 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 4.8.1).
- * REQ28: Specify and register the identifier of newly defined semantics for binary scopes (see Section 7).
- * REQ29: Categorize newly defined parameters according to the same criteria of Section 8.

- * REQ30: Define whether Clients must, should or may support the conditional parameters defined in Section 8, and under which circumstances.

A.2. Optional-to-Address Requirements

- * OPT1: Optionally, if the textual format of 'scope' is used, specify CBOR values to use for abbreviating the role identifiers in the group (see Section 3.1).
- * OPT2: Optionally, specify the additional parameters used in the exchange of Token Transfer Request and Response (see Section 3.3).
- * OPT3: Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' is not used (see Section 3.3).
- * OPT4: Optionally, specify possible or required payload formats for specific error cases.
- * OPT5: Optionally, specify additional identifiers of error types, as values of the 'error' field in an error response from the KDC.
- * OPT6: Optionally, specify the encoding of 'pub_keys_repos' if the default is not used (see Section 4.3.1).
- * OPT7: Optionally, specify the functionalities implemented at the 'control_uri' resource hosted at the Client, including message exchange encoding and other details (see Section 4.3.1).
- * OPT8: Optionally, specify the behavior of the handler in case of failure to retrieve a public key for the specific node (see Section 4.3.1).
- * OPT9: Optionally, define a default group rekeying scheme, to refer to in case the 'rekeying_scheme' parameter is not included in the Joining Response (see Section 4.3.1).
- * OPT10: Optionally, specify the functionalities implemented at the 'control_group_uri' resource hosted at the Client, including message exchange encoding and other details (see Section 4.3.1).
- * OPT11: Optionally, specify policies that instruct Clients to retain messages and for how long, if they are unsuccessfully decrypted (see Section 4.8.1.1). This makes it possible to decrypt such messages after getting updated keying material.

- * OPT12: 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 4.8.2.1).
- * OPT13: Optionally, specify how the identifier of a group members's public key is included in requests sent to other group members (see Section 4.9.1.1).
- * OPT14: Optionally, specify additional information to include in rekeying messages for the "Point-to-Point" group rekeying scheme (see Section 6).
- * OPT15: Optionally, specify if Clients must or should support any of the parameters defined as optional in this specification (see Section 8).

Appendix B. Extensibility for Future COSE Algorithms

As defined in Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs], future algorithms can be registered in the "COSE Algorithms" registry [COSE.Algorithms] as specifying none or multiple COSE capabilities.

To enable the seamless use of such future registered algorithms, this section defines a general, agile format for each 'sign_info_entry' of the 'sign_info' parameter in the Token Transfer Response, see Section 3.3.1.

If any of the currently registered COSE algorithms is considered, using this general format yields the same structure of 'sign_info_entry' defined in this document, thus ensuring retro-compatibility.

B.1. Format of 'sign_info_entry'

The format of each 'sign_info_entry' (see Section 3.3.1) is generalized as follows. Given N the number of elements of the 'sign_parameters' array, i.e., the number of COSE capabilities of the signature algorithm, then:

- * 'sign_key_parameters' is replaced by N elements 'sign_capab_i', each of which is a CBOR array.
- * The i-th array following 'sign_parameters', i.e., 'sign_capab_i' (i = 0, ..., N-1), is the array of COSE capabilities for the algorithm capability specified in 'sign_parameters'[i].

```
sign_info_entry =  
[  
  id : gname / [ + gname ],  
  sign_alg : int / tstr,  
  sign_parameters : [ alg_capab_1 : any,  
                      alg_capab_2 : any,  
                      ...,  
                      alg_capab_N : any],  
  sign_capab_1 : [ any ],  
  sign_capab_2 : [ any ],  
  ...,  
  sign_capab_N : [ any ],  
  pub_key_enc = int / nil  
]  
  
gname = tstr
```

Figure 37: 'sign_info_entry' with general format

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. Version -14 to -15

- * Fixed nits.

C.2. Version -13 to -14

- * Clarified scope and goal of the document in abstract and introduction.
- * Overall clarifications on semantics of operations and parameters.
- * Major restructuring in the presentation of the KDC interface.
- * Revised error handling, also removing redundant text.
- * Imported parameters and KDC resource about the KDC's public key from draft-ietf-ace-key-groupcomm-oscore.
- * New parameters 'group_rekeying_scheme' and 'control_group_uri'.
- * Provided example of administrative keying material transported in 'mgt_key_material'.
- * Reasoned categorization of parameters, as expected support by ACE Clients.

- * Reasoned categorization of KDC functionalities, as minimally/optional to support for ACE Clients.
- * Guidelines on enhanced error responses using 'error' and 'error_description'.
- * New section on group rekeying, discussing at a high-level a basic one-to-one approach and possible one-to-many approaches.
- * Revised and expanded security considerations, also about the KDC.
- * Updated list of requirements for application profiles.
- * Several further clarifications and editorial improvements.

C.3. Version -05 to -13

- * Incremental revision of the KDC interface.
- * Removed redundancy in parameters about signature algorithm and signature keys.
- * Node identifiers always indicated with 'peer_identifiers'.
- * Format of public keys changed from raw COSE Keys to be certificates, CWTs or CWT Claims Set (CCS). Adapted parameter 'pub_key_enc'.
- * Parameters and functionalities imported from draft-ietf-key-groupcomm-oscore where early defined.
- * Possible provisioning of the KDC's Diffie-Hellman public key in response to the Token transferring to /authz-info.
- * Generalized proof-of-possession evidence, to be not necessarily a signature.
- * Public keys of group members may be retrieved filtering by role and/or node identifier.
- * Enhanced error handling with error code and error description.
- * Extended "typed" format for the 'scope' claim, optional to use.
- * Editorial improvements.

C.4. Version -04 to -05

- * Updated uppercase/lowercase URI segments for KDC resources.
- * Supporting single Access Token for multiple groups/topics.
- * Added 'control_uri' 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 OPT11.
- * 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 OPT12.
- * Editorial improvements.

C.5. 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.

C.6. Version -02 to -03

- * Exchange of information on the signature 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).

C.7. 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).

C.8. 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, Watson Ladd, John Mattsson, Daniel Migault, Jim Schaad, Ludwig Seitz, Goeran Selander, Cigdem Sengul 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 7, 2020

F. Palombini
Ericsson
November 04, 2019

CoAP Pub-Sub Profile for Authentication and Authorization for
Constrained Environments (ACE)
draft-palombini-ace-coap-pubsub-profile-06

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.

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 May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Application Profile Overview	3
3. coap_pubsub_app Application Profile	5
3.1. Retrieval of COSE Key for protection of content	5
4. Publisher	8
5. Subscriber	10
6. Pub-Sub Protected Communication	12
6.1. Using COSE Objects To Protect The Resource Representation	13
7. Security Considerations	14
8. IANA Considerations	15
8.1. ACE Groupcomm Profile Registry	15
8.2. ACE Groupcomm Key Registry	16
9. References	16
9.1. Normative References	16
9.2. Informative References	17
Appendix A. Requirements on Application Profiles	17
Acknowledgments	19
Author's Address	19

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]. 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.

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] and [I-D.ietf-core-coap-pubsub]. 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].

2. Application Profile Overview

The objective of this document is to specify how to authorize nodes, provide keys, and protect a CoAP pub-sub communication, as described in [I-D.ietf-core-coap-pubsub], 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]).

The architecture of the scenario is shown in Figure 1.

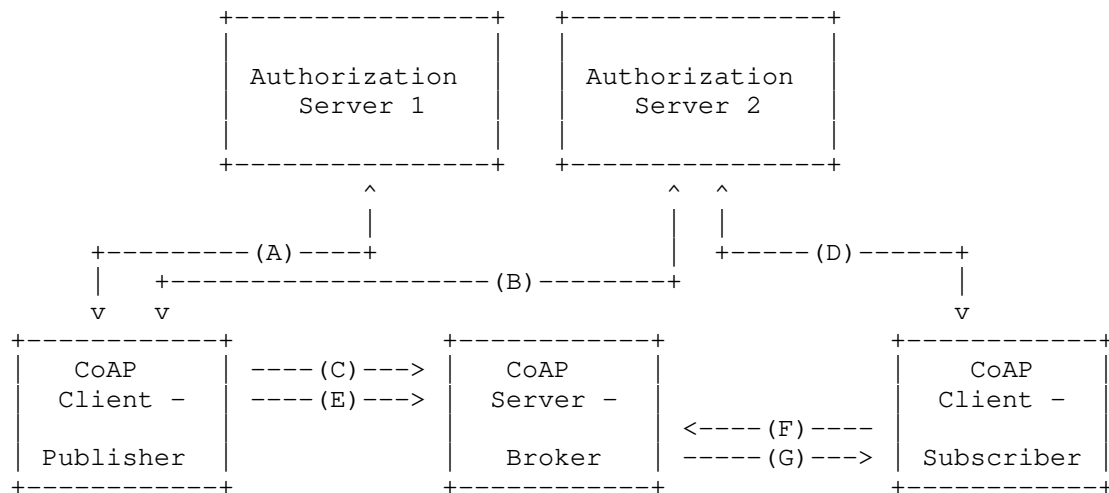


Figure 1: Architecture CoAP pubsub with Authorization Servers

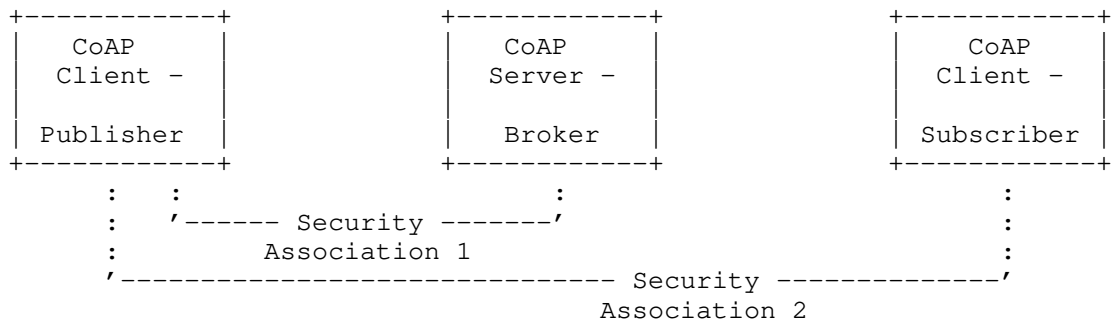
The RS is the broker, which contains the topic. This node corresponds to the Dispatcher, in [I-D.ietf-ace-key-groupcomm]. The AS1 hosts the policies about the Broker: what endpoints are allowed to Publish on the Broker. The Clients access this node to get write access to the Broker. The AS2 hosts the 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. coap_pubsub_app Application Profile

This profile uses [I-D.ietf-ace-key-groupcomm], which expands the ACE framework. This document specifies which exact parameters from [I-D.ietf-ace-key-groupcomm] have to be used, and the values for each parameter.

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, called "coap_pubsub_app".

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.

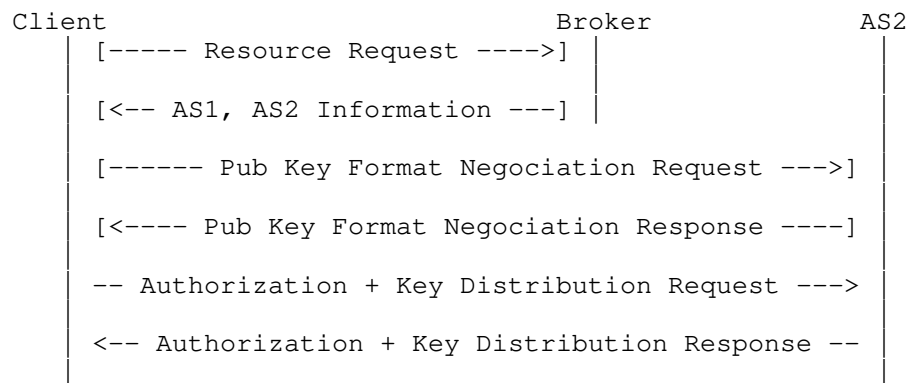


Figure 2: B: Access request - response

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 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):

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

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:

- o the following fields from the Joining Response (Section 4.1 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:
 - + 'kty' with value 4 (symmetric)
 - + 'alg' with value defined by the AS2 (Content Encryption Algorithm)

- + 'Base IV' with value defined by the AS2
- + 'k' with value the symmetric key value
- + 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
- o the following fields from the Authorization Response (Section 3.2 of [I-D.ietf-ace-key-groupcomm]):
 - * 'profile' set to "coap_pubsub_app", as specified in Section 8.1
 - * OPTIONALLY 'scope', set to a CBOR array containing:
 - + the broker's topic as first element, and
 - + 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.

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.

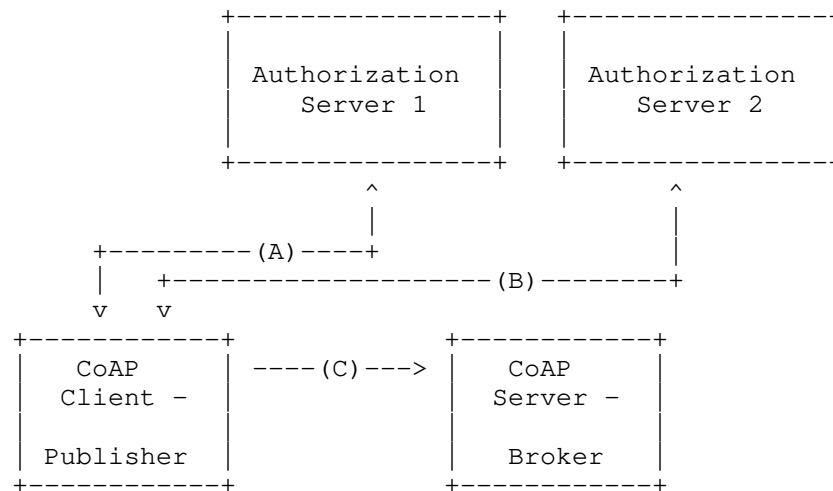


Figure 4: Phase 1: Publisher side

This is a combination of two independent phases:

- o 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] or OSCORE [I-D.ietf-ace-oscore-profile]. (A) (C)
- o 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.

(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.

An example of the payload of an Authorization + Joining Request and corresponding Response for a Publisher 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'd36b581dleef9c7c,
      "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

5. Subscriber

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

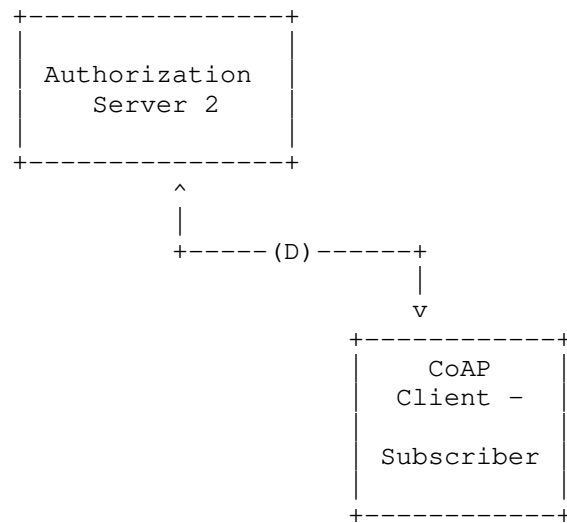


Figure 7: Phase 2: Subscriber side

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:

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

An example of the payload of an Authorization + Joining Request and corresponding Response for a Subscriber 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

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].

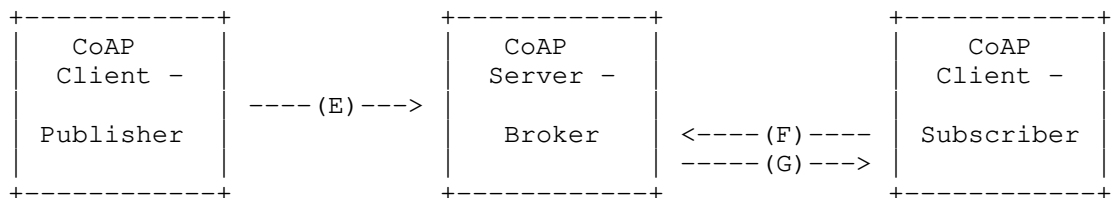


Figure 10: Phase 3: Secure communication between Publisher and Subscriber

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.

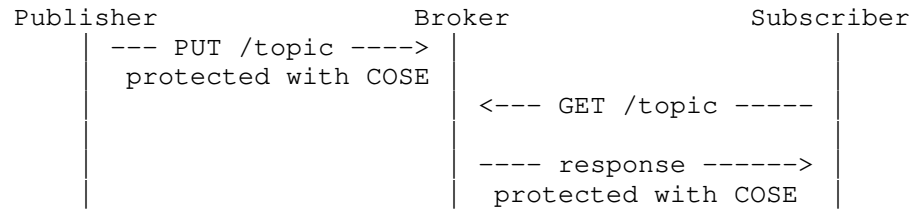


Figure 11: (E), (F), (G): Example of protected communication

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]). 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:

- o 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.
- o 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].
- o 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.

Name: coap_pubsub_app

Description: Profile for delegating client authentication and authorization for publishers and subscribers in a 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", draft-ietf-ace-key-groupcomm-03 (work in progress), November 2019.
- [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-25 (work in progress), October 2019.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-09 (work in progress), September 2019.
- [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>>.
- [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", draft-ietf-ace-actors-07 (work in progress), October 2018.
- [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-08 (work in progress), April 2019.
- [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-08 (work in progress), July 2019.

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]

- o REQ1: Specify the encoding and value of the identifier of group or topic of 'scope': see Section 3.1).
- o REQ2: Specify the encoding and value of roles of 'scope': see Section 3.1).
- o REQ3: Optionally, specify the acceptable values for 'sign_alg': TODO
- o REQ4: Optionally, specify the acceptable values for 'sign_parameters': TODO
- o REQ5: Optionally, specify the acceptable values for 'sign_key_parameters': TODO

- o REQ6: Optionally, specify the acceptable values for 'pub_key_enc': TODO
- o REQ7: Specify the exact format of the 'key' value: COSE_Key, see Section 3.1.
- o REQ8: Specify the acceptable values of 'kty' : "COSE_Key", see Section 3.1.
- o REQ9: Specify the format of the identifiers of group members: TODO
- o REQ10: Optionally, specify the format and content of 'group_policies' entries: not defined
- o REQ11: Specify the communication protocol the members of the group must use: CoAP pub/sub.
- o 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.
- o REQ13: Specify and register the application profile identifier : "coap_pubsub_app", see Section 8.1.
- o REQ14: Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: NA.
- o REQ15: Specify policies at the KDC to handle id that are not included in get_pub_keys: TODO
- o REQ16: Specify the format and content of 'group_policies': TODO
- o 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
- o REQ18: Specify how the communication is secured between Client and KDC. Optionally, specify transport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and KDC: pre-set, as KDC is AS.
- o OPT1: Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: NA
- o 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

- o OPT3: Optionally, specify the format and content of 'mgt_key_material': not defined
- o 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, Jim Schaad and Marco Tiloca for the useful discussion and reviews that helped shape this document.

Author's Address

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

M. Tiloca
R. Höglund
RISE AB
L. Seitz
Combitech
F. Palombini
Ericsson AB
7 March 2022

Group OSCORE Profile of the Authentication and Authorization for
Constrained Environments Framework
draft-tiloca-ace-group-oscure-profile-08

Abstract

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework. The profile uses Group OSCORE to provide communication security between a Client and a (set of) Resource Server(s) as members of an OSCORE Group. The profile securely binds an OAuth 2.0 Access Token with the public key of the Client associated with the private key used in the OSCORE group. The profile uses Group OSCORE to achieve server authentication, as well as proof-of-possession for the Client's public key. Also, it provides proof of the Client's membership to the correct OSCORE group, by binding the Access Token to information from the Group OSCORE Security Context, thus allowing the Resource Server(s) to verify the Client's membership upon receiving a message protected with Group OSCORE from the Client.

Discussion Venues

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

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (ace@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/ace/>.

Source for this draft and an issue tracker can be found at <https://gitlab.com/crimson84/draft-tiloca-ace-group-oscure-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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
1.1. Terminology	6
2. Protocol Overview	8
2.1. Pre-Conditions	9
2.2. Access Token Retrieval	10
2.3. Access Token Posting	11
2.4. Secure Communication	11
3. Client-AS Communication	12
3.1. C-to-AS: POST to Token Endpoint	12
3.1.1. 'context_id' Parameter	15
3.1.2. 'salt_input' Parameter	15
3.1.3. 'client_cred_verify' Parameter	16
3.1.4. 'client_cred_verify_mac' Parameter	16
3.2. AS-to-C: Access Token	16
3.2.1. Salt Input Claim	20
3.2.2. Context ID Input Claim	21
4. Client-RS Communication	21
4.1. C-to-RS POST to authz-info Endpoint	22
4.2. RS-to-C: 2.01 (Created)	22
4.3. Client-RS Secure Communication	24

4.3.1. Client Side	24
4.3.2. Resource Server Side	24
4.4. Access Rights Verification	25
4.5. Change of Client's Authentication Credential in the Group	25
5. Secure Communication with the AS	26
6. Discarding the Security Context	26
7. CBOR Mappings	27
8. Security Considerations	27
9. Privacy Considerations	28
10. IANA Considerations	29
10.1. ACE Profile Registry	29
10.2. OAuth Parameters Registry	29
10.3. OAuth Parameters CBOR Mappings Registry	31
10.4. CBOR Web Token Claims Registry	32
10.5. TLS Exporter Label Registry	33
11. References	33
11.1. Normative References	33
11.2. Informative References	36
Appendix A. Dual Mode (Group OSCORE & OSCORE)	37
A.1. Protocol Overview	38
A.1.1. Pre-Conditions	40
A.1.2. Access Token Posting	40
A.1.3. Setup of the Pairwise OSCORE Security Context	41
A.1.4. Secure Communication	42
A.2. Client-AS Communication	42
A.2.1. C-to-AS: POST to Token Endpoint	43
A.2.2. AS-to-C: Access Token	45
A.3. Client-RS Communication	53
A.3.1. C-to-RS POST to authz-info Endpoint	54
A.3.2. RS-to-C: 2.01 (Created)	54
A.3.3. OSCORE Setup - Client Side	56
A.3.4. OSCORE Setup - Resource Server Side	58
A.3.5. Access Rights Verification	61
A.3.6. Change of Client's Authentication Credential in the Group	61
A.4. Secure Communication with the AS	62
A.5. Discarding the Security Context	62
A.6. CBOR Mappings	63
A.7. Security Considerations	63
A.8. Privacy Considerations	64
Appendix B. Profile Requirements	64
Acknowledgments	65
Authors' Addresses	65

1. Introduction

A number of applications rely on a group communication model, where a Client can access a resource shared by multiple Resource Servers at once, e.g., over IP multicast. Typical examples are switching of luminaries, actuators control, and distribution of software updates. Secure communication in the group can be achieved by sharing a set of keying material, which is typically provided upon joining the group.

For some of such applications, it may be just fine to enforce access control in a straightforward fashion. That is, any Client authorized to join the group, hence to get the group keying material, can be also implicitly authorized to perform any action at any resource of any Server in the group. An example of application where such implicit authorization might be used is a simple lighting scenario, where the lightbulbs are the Servers, while the user account on an app on the user's phone is the Client. In this case, it might be fine to not require additional authorization evidence from any user account, if it is acceptable that any current group member is also authorized to switch on and off any light, or to check their status.

However, in different instances of such applications, the approach above is not desirable, as different group members are intended to have different access rights to resources of other group members. That is, access control to the secure group communication channel and access control to the resource space provided by servers in the group should remain logically separated domains. For instance, a more fine-grained approach is required in the two following use cases.

As a first case, an application provides control of smart locks acting as Servers in the group, where: a first type of Client, e.g., a user account of a child, is allowed to only query the status of the smart locks; while a second type of Client, e.g., a user account of a parent, is allowed to both query and change the status of the smart locks. Further similar applications concern the enforcement of different sets of permissions in groups with sensor/actuator devices, e.g., thermostats, acting as Servers. Also, some group members may even be intended as Servers only. Hence, they must be prevented from acting as Clients altogether and from accessing resources at other Servers, especially when attempting to perform non-safe operations.

As a second case, building automation scenarios often rely on Servers that, under different circumstances, enforce different level of priority for processing received commands. For instance, BACnet deployments consider multiple classes of Clients, e.g., a normal light switch (C1) and an emergency fire panel (C2). Then, a C1 Client is not allowed to override a command from a C2 Client, until the latter relinquishes control at its higher priority. That is: i)

only C2 Clients should be able to adjust the minimum required level of priority on the Servers, so rightly locking out C1 Clients if needed; and ii) when a Server is set to accept only high-priority commands, only C2 Clients should be able to perform such commands otherwise allowed also to C1 Clients. Given the different maximum authority of different Clients, fine-grained access control would effectively limit the execution of high- and emergency-priority commands only to devices that are in fact authorized to do so. Besides, it would prevent a misconfigured or compromised device from initiating a high-priority command and lock out normal control.

In the cases above, being a legitimate group member and storing the group keying material is not supposed to imply any particular access rights. Also, introducing a different security group for each different set of access rights would result in additional keying material to distribute and manage. In particular, if the access rights for a single node change, this would require to evict that node from the current group, followed by that node joining a different group aligned with its new access rights. Moreover, the keying material of both groups would have to be renewed for their current members. Overall, this would have a non negligible impact on operations and performance in the system.

A fine-grained access control model can be rather enforced within a same group, by using the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. That is, a Client has to first obtain authorization credentials in the form of an Access Token, and post it to the Resource Server(s) in the group before accessing the intended resources.

The ACE framework delegates to separate profile documents how to secure communications between the Client and the Resource Server. However each of the current profiles of ACE defined in [I-D.ietf-ace-oscore-profile] [I-D.ietf-ace-dtls-authorize] [I-D.ietf-ace-mqtt-tls-profile] admits a single security protocol that cannot be used to protect group messages sent over IP multicast.

This document specifies the "coap_group_oscore" profile of the ACE framework, where a Client uses CoAP [RFC7252] or CoAP over IP multicast [I-D.ietf-core-groupcomm-bis] to communicate to one or multiple Resource Servers, which are members of an application group and share a common set of resources. This profile uses Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the security protocol to protect messages exchanged between the Client and the Resource Servers. Hence, it requires that both the Client and the Resource Servers have previously joined the same OSCORE group.

That is, this profile describes how access control is enforced for a Client after it has joined an OSCORE group, to access resources at other members in that group. The process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] takes place before the process described in this document, and is out of the scope of this profile.

The Client proves its access to be authorized to the Resource Server by using an Access Token, which is bound to a key (the proof-of-possession key). This profile uses Group OSCORE to achieve server authentication, as well as proof-of-possession for the Client's public key used in the OSCORE group in question. Note that the proof of possession is not done by a dedicated protocol element, but rather occurs after the first Group OSCORE exchange.

Furthermore, this profile provides proof of the Client's membership to the correct OSCORE group, by binding the Access Token to the Client's authentication credential used in the group and including the Client's public key, as well as to information from the pre-established Group OSCORE Security Context. This allows the Resource Server to verify the Client's group membership upon reception of a message protected with Group OSCORE from that Client.

OSCORE [RFC8613] specifies how to use COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] to secure CoAP messages. Group OSCORE builds on OSCORE to provide secure group communication, and ensures source authentication: by means of digital signatures embedded in protected messages (in group mode); or by protecting messages with pairwise keying material derived from the asymmetric keys of the two peers exchanging the message (in pairwise mode).

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts related to CBOR [RFC8949], COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], CoAP [RFC7252], OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm]. These especially include:

- * Group Manager, as the entity responsible for a set of groups where communications among members are secured with Group OSCORE.

- * Authentication credential, as the set of information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert].

Members of an OSCORE group have an associated authentication credential in the format used in the group. As per Section 2.3 of [I-D.ietf-core-oscore-groupcomm], an authentication credential provides the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

Readers are expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz], as well as in the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

Additionally, this document makes use of the following terminology.

- * Equivalent COSE Key: a COSE Key built from an authentication credential used in the OSCORE group. The equivalent COSE Key preserves all the main information elements from the authentication credential, in particular the key coordinates and the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).
- * Pairwise-only group: an OSCORE group that uses only the pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

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

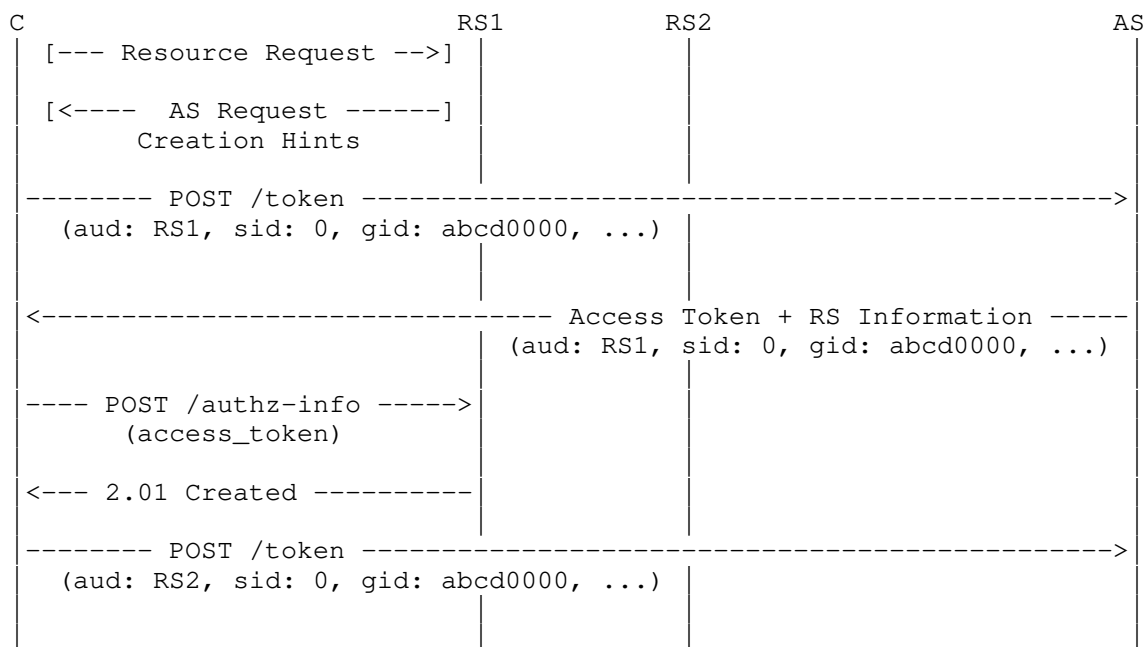
2. Protocol Overview

This section provides an overview of this profile, i.e., on how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to secure communications between a Client and a (set of) Resource Server(s) using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Note that this profile of ACE describes how access control can be enforced for a node after it has joined an OSCORE group, to access resources at other members in that group.

In particular, the process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] must take place before the process described in this document, and is out of the scope of this profile.

An overview of the protocol flow for this profile is shown in Figure 1. In the figure, it is assumed that both RS1 and RS2 are associated with the same AS. It is also assumed that C, RS1 and RS2 have previously joined an OSCORE group with Group Identifier (gid) "abcd0000", and got assigned Sender ID (sid) "0", "1" and "2" in the group, respectively. The names of messages coincide with those of [I-D.ietf-ace-oauth-authz] when applicable.



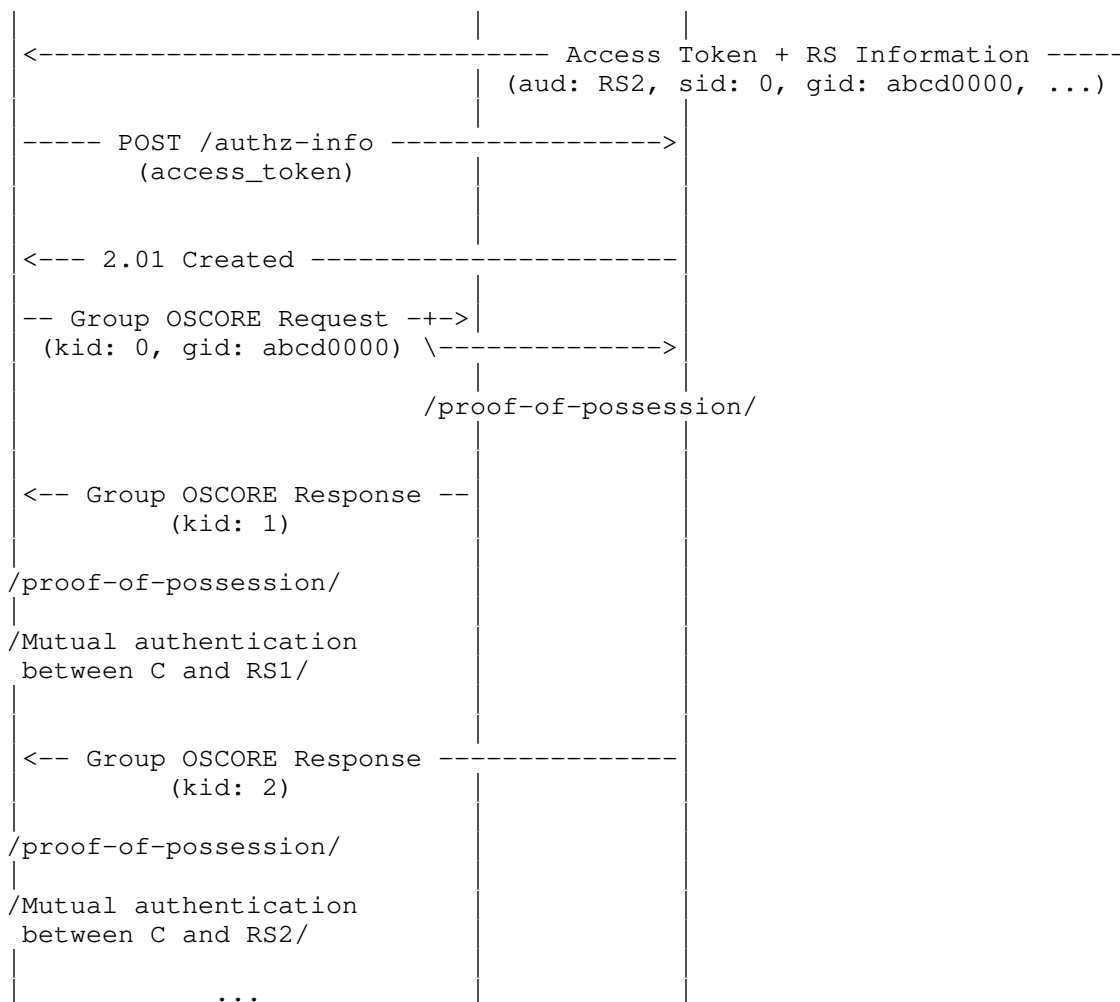


Figure 1: Protocol Overview.

2.1. Pre-Conditions

Using Group OSCORE and this profile requires both the Client and the Resource Servers to have previously joined the same OSCORE group. This especially includes the derivation of the Group OSCORE Security Context and the assignment of unique Sender IDs to use in the group. Nodes may join the OSCORE group through the respective Group Manager by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscure], which is also based on ACE.

After the Client and Resource Servers have joined the group, this profile provides access control for accessing resources on those Resource Servers, by securely communicating with Group OSCORE.

As a pre-requisite for this profile, the Client has to have successfully joined the OSCORE group where also the Resource Servers (RSs) are members. Depending on the limited information initially available, the Client may have to first discover the exact OSCORE group used by the RSs for the resources of interest, e.g., by using the approach defined in [I-D.tiloca-core-oscore-discovery].

2.2. Access Token Retrieval

This profile requires that the Client retrieves an Access Token from the AS for the resource(s) it wants to access on each of the RSs, using the /token endpoint, as specified in Section 5.8 of [I-D.ietf-ace-oauth-authz]. In a general case, it can be assumed that different RSs are associated with different ASs, even if the RSs are members of a same OSCORE group.

In the Access Token request to the AS, the Client MUST include the Group Identifier of the OSCORE group and its own Sender ID in that group. The AS MUST specify these pieces of information in the Access Token, included in the Access Token response to the Client.

Furthermore, in the Access Token request to the AS, the Client MUST also include: its own public key used in the OSCORE group; and a proof-of-possession (PoP) evidence to proof possession of the corresponding private key. The PoP evidence is computed over a PoP input uniquely related to the secure communication association between the Client and the AS. The AS MUST include also the public key indicated by the Client in the Access Token.

The Access Token request and response MUST be confidentiality-protected and ensure authenticity. This profile RECOMMENDS the use of OSCORE between the Client and the AS, to reduce the number of libraries the client has to support. Other protocols fulfilling the security requirements defined in Sections 5 and 6 of [I-D.ietf-ace-oauth-authz] MAY alternatively be used, such as TLS [RFC8446] or DTLS [RFC6347][I-D.ietf-tls-dtls13].

2.3. Access Token Posting

After having retrieved the Access Token from the AS, the Client posts the Access Token to the RS, using the /authz-info endpoint and mechanisms specified in Section 5.10 of [I-D.ietf-ace-oauth-authz], as well as Content-Format = application/ace+cbor. When using this profile, the communication with the /authz-info endpoint is not protected.

If the Access Token is valid, the RS replies to this POST request with a 2.01 (Created) response with Content-Format = application/ace+cbor. Also, the RS associates the received Access Token with the Group OSCORE Security Context identified by the Group Identifier specified in the Access Token, following Section 3.2 of [RFC8613]. In practice, the RS maintains a collection of Security Contexts with associated authorization information, for all the clients that it is currently communicating with, and the authorization information is a policy used as input when processing requests from those clients.

Finally, the RS stores the association between i) the authorization information from the Access Token; and ii) the Group Identifier of the OSCORE group together with the Sender ID and the authentication credential of the Client in that group. This binds the Access Token with the Group OSCORE Security Context of the OSCORE group.

Finally, when the Client communicates with the RS using the Group OSCORE Security Context, the RS verifies that the Client is a legitimate member of the OSCORE group and especially the exact group member with the same Sender ID associated with the Access Token. This occurs when verifying a request protected with Group OSCORE, since the request includes the Client's Sender ID and either it embeds a signature computed also over that Sender ID (if protected with the group mode), or it is protected by means of pairwise symmetric keying material derived from the asymmetric keys of the two peers (if protected with the pairwise mode).

2.4. Secure Communication

The Client can send a request protected with Group OSCORE [I-D.ietf-core-oscore-groupcomm] to the RS. This can be a unicast request addressed to the RS, or a multicast request addressed to the OSCORE group where the RS is also a member. To this end, the Client uses the Group OSCORE Security Context already established upon joining the OSCORE group, e.g., by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. The RS may send a response back to the Client, protecting it by means of the same Group OSCORE Security Context.

3. Client-AS Communication

This section details the Access Token POST Request that the Client sends to the /token endpoint of the AS, as well as the related Access Token response.

The Access Token MUST be bound to the public key of the client as proof-of-possession key (pop-key), by means of the 'cnf' claim.

3.1. C-to-AS: POST to Token Endpoint

The Client-to-AS request is specified in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. The Client MUST send this POST request to the /token endpoint over a secure channel that guarantees authentication, message integrity and confidentiality.

The POST request is formatted as the analogous Client-to-AS request in the OSCORE profile of ACE (see Section 3.1 of [I-D.ietf-ace-oscore-profile]), with the following additional parameters that MUST be included in the payload.

- * 'context_id', defined in Section 3.1.1 of this document. This parameter specifies the Group Identifier (GID), i.e., the Id Context of an OSCORE group where the Client and the RS are currently members. In particular, the Client wishes to communicate with the RS using the Group OSCORE Security Context associated with that OSCORE group.
- * 'salt_input', defined in Section 3.1.2 of this document. This parameter includes the Sender ID that the Client has in the OSCORE group whose GID is specified in the 'context_id' parameter above.
- * 'req_cnf', defined in Section 3.1 of [I-D.ietf-ace-oauth-params]. This parameter follows the syntax from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. In particular, the specified public key is the COSE Key equivalent to the authentication credential that the Client uses in the OSCORE group. The specified public key will be used as the pop-key bound to the Access Token.

Alternative Value Types defined in future specifications are fine to consider, if indicating a non-encrypted asymmetric key or full-fledged authentication credential.

In addition, the Client computes its proof-of-possession (PoP) evidence, in order to prove possession of its own private key used in the OSCORE group to the AS. This allows the AS to verify that the Client indeed owns the private key associated with that public key, as its alleged identity credential within the OSCORE group.

To this end, the Client MUST use as PoP input the byte representation of a quantity that uniquely represents the secure communication association between the Client and the AS. It is RECOMMENDED that the Client considers the following as PoP input.

- * If the Client and the AS communicate over (D)TLS, the PoP input is an exporter value computed as defined in Section 7.5 of [RFC8446]. In particular, the exporter label MUST be 'EXPORTER-ACE-Sign-Challenge-Client-AS' defined in Section 10.5 of this document, together with an empty 'context_value', and 32 bytes as 'key_length'.
- * If the Client and the AS communicate over OSCORE, the PoP input is the output PRK of a HKDF-Extract step [RFC5869], i.e., $PRK = \text{HMAC-Hash}(\text{salt}, \text{IKM})$. In particular, 'salt' takes $(x1 \parallel x2)$, where $x1$ is the ID Context of the OSCORE Security Context between the Client and the AS, $x2$ is the Sender ID of the Client in that Security Context, and \parallel denotes byte string concatenation. Also, 'IKM' is the OSCORE Master Secret of the OSCORE Security Context between the Client and the AS.

The HKDF MUST be one of the HMAC-based HKDF [RFC5869] algorithms defined for COSE [I-D.ietf-cose-rfc8152bis-algs]. The Client and AS may agree on the HKDF algorithm to use during the Client's registration at the AS. HKDF SHA-256 is mandatory to implement.

Then, the Client computes the PoP evidence as follows.

- * If the OSCORE group is not a pairwise-only group, the PoP evidence MUST be a signature. The Client computes the signature by using the same private key and signature algorithm it uses for signing messages in the OSCORE group. The private key corresponds to the authentication credential used in the OSCORE group, for which the equivalent COSE Key is specified in the 'req_cnf' parameter above.
- * If the OSCORE group is a pairwise-only group, the PoP evidence MUST be a MAC computed as follows, by using the HKDF Algorithm HKDF SHA-256, which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

$MAC = \text{HKDF}(\text{salt}, \text{IKM}, \text{info}, L)$

The input parameters of HKDF are as follows.

- salt takes as value the empty byte string.
- IKM is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using an ECDH algorithm pre-agreed between Client and AS. The Client uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the AS. For X25519 and X448, the procedure is described in Section 5 of [RFC7748].

The Client's private key corresponds to the Client's authentication credential used in the OSCORE group, for which the equivalent COSE Key is specified in the 'req_cnf' parameter above. The Client may obtain the Diffie-Hellman public key of the AS during its registration process at the AS.

The Client and AS may agree on the ECDH algorithm to use during the Client's registration at the AS. The ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement.

- info takes as value the PoP input.
- L is equal to 8, i.e., the size of the MAC, in bytes.

Finally, the Client MUST include one of the two following parameters in the payload of the POST request to the AS.

- * 'client_cred_verify', defined in Section 3.1.3 of this document, specifying the Client's PoP evidence as a signature, which is computed as defined above. This parameter MUST be included if and only if the OSCORE group is not a pairwise-only group.
- * 'client_cred_verify_mac', defined in Section 3.1.4 of this document, specifying the Client's PoP evidence as a MAC, which is computed as defined above. This parameter MUST be included if and only if the OSCORE group is a pairwise-only group.

An example of such a request is shown in Figure 2.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "audience" : "tempSensor4711",
  "scope" : "read",
  "context_id" : h'abcd0000',
  "salt_input" : h'00',
  "req_cnf" : {
    "COSE_Key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
        27c9e354089bbe13',
      "y" : h'f95eld4b851a2cc80fff87d8e23f22afb725d535e515d020
        731e79a3b4e47120'
    }
  },
  "client_cred_verify" : h'...'
  (signature content omitted for brevity)
}
```

Figure 2: Example C-to-AS POST /token request for an Access Token bound to an asymmetric key.

3.1.1. 'context_id' Parameter

The 'context_id' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. This parameter provides a value that the Client wishes to use with the RS as a hint for a security context. Its exact content is profile specific.

3.1.2. 'salt_input' Parameter

The 'salt_input' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. This parameter provides a value that the Client wishes to use as part of a salt with the RS, for deriving cryptographic keying material. Its exact content is profile specific.

3.1.3. 'client_cred_verify' Parameter

The 'client_cred_verify' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.8.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides a signature computed by the Client to prove the possession of its own private key.

3.1.4. 'client_cred_verify_mac' Parameter

The 'client_cred_verify_mac' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.8.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides a Message Authentication Code (MAC) computed by the Client to prove the possession of its own private key.

3.2. AS-to-C: Access Token

After having verified the POST request to the /token endpoint and that the Client is authorized to obtain an Access Token corresponding to its Access Token request, the AS MUST verify the proof-of-possession (PoP) evidence. In particular, the AS proceeds as follows.

- * As PoP input, the AS uses the same value considered by the Client in Section 3.1.
- * As public key of the Client, the AS uses the one specified in the 'req_cnf' parameter of the Access Token request.
- * If the Access Token request includes the 'client_cred_verify' parameter, this specifies the PoP evidence as a signature. Then, the AS verifies the signature by using the public key of the Client.
- * If the Access Token request includes the 'client_cred_verify_mac' parameter, this specifies the PoP evidence as a Message Authentication Code (MAC).

Then, the AS recomputes the MAC through the same process taken by the Client when preparing the value of the 'client_cred_verify_mac' parameter for the Access Token (see Section 3.1), with the difference that the AS uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the Client. The verification succeeds if and only if the recomputed MAC is equal to the MAC conveyed as PoP evidence in the Access Token request.

If both the 'client_cred_verify' and 'client_cred_verify_mac' parameters are present, or if the verification of the PoP evidence fails, the AS considers the Client request invalid.

If the Client request was invalid, or not authorized, the AS returns an error response as described in Section 5.8.3 of [I-D.ietf-ace-oauth-authz].

If all verifications are successful, the AS responds as defined in Section 5.8.2 of [I-D.ietf-ace-oauth-authz]. In particular:

- * The AS can signal that the use of Group OSCORE is REQUIRED for a specific Access Token by including the 'ace_profile' parameter with the value "coap_group_oscore" in the Access Token response. The Client MUST use Group OSCORE towards all the Resource Servers for which this Access Token is valid. Usually, it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile signaling can be omitted.
- * The AS MUST NOT include the 'rs_cnf' parameter defined in [I-D.ietf-ace-oauth-params]. In general, the AS may not be aware of the authentication credentials (and public keys included thereof) that the RSs use in the OSCORE group. Also, the Client is able to retrieve the authentication credentials of other group members from the responsible Group Manager, both upon joining the group or later on as a group member, as defined in [I-D.ietf-ace-key-groupcomm-oscore].

The AS MUST include the following information as metadata of the issued Access Token. The use of CBOR web tokens (CWT) as specified in [RFC8392] is RECOMMENDED.

- * The profile "coap_group_oscore". If the Access Token is a CWT, this is placed in the 'ace_profile' claim of the Access Token, as per Section 5.10 of [I-D.ietf-ace-oauth-authz].
- * The salt input specified in the 'salt_input' parameter of the Token Request. If the Access Token is a CWT, the content of the 'salt_input' parameter MUST be placed in the 'salt_input' claim of the Access Token, defined in Section 3.2.1 of this document.
- * The Context Id input specified in the 'context_id' parameter of the Token Request. If the Access Token is a CWT, the content of the 'context_id' parameter MUST be placed in the 'contextId_input' claim of the Access Token, defined in Section 3.2.2 of this document.

- * The public key that the client uses in the OSCORE group and specified in the 'req_cnf' parameter of the Token request.

If the Access Token is a CWT, the public key MUST be specified in the 'cnf' claim, which follows the syntax from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. In particular, the 'cnf' claim includes the same COSE Key specified in the 'req_cnf' parameter of the Token Request, i.e., the COSE Key equivalent to the authentication credential that the Client uses in the OSCORE group.

Alternative Value Types defined in future specifications are fine to consider, if indicating a non-encrypted asymmetric key or full-fledged authentication credential.

Figure 3 shows an example of such an AS response. The access token has been truncated for readability.

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

Figure 3: Example AS-to-C Access Token response with the Group OSCORE profile.

Figure 4 shows an example CWT Claims Set, containing the Client's public key in the group (as pop-key) in the 'cnf' claim.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
        27c9e354089bbe13',
      "y" : h'f95eld4b851a2cc80fff87d8e23f22afb725d535e515d020
        731e79a3b4e47120'
    },
    "salt_input" : h'00',
    "contextId_input" : h'abcd0000'
  }
}
```

Figure 4: Example CWT Claims Set with OSCORE parameters.

The same CWT Claims Set as in Figure 4 and encoded in CBOR is shown in Figure 5, using the value abbreviations defined in [I-D.ietf-ace-oauth-authz] and [RFC8747]. The bytes in hexadecimal are reported in the first column, while their corresponding CBOR meaning is reported after the "#" sign on the second column, for easiness of readability.

NOTE: it should be checked (and in case fixed) that the values used below (which are not yet registered) are the final values registered by IANA.


```

A7                                     # map(7)
  03                                 # unsigned(3)
  76                                 # text(22)
    74656D7053656E736F72496E4C6976696E67526F6F6D
  06                                 # unsigned(6)
  1A 5112D728                       # unsigned(1360189224)
  04                                 # unsigned(4)
  1A 51145DC8                       # unsigned(1360289224)
  09                                 # unsigned(9)
  78 18                             # text(24)
    74656D70657261747572655F67206669726D776172655F70
  08                                 # unsigned(8)
  A1                                 # map(1)
    01                             # unsigned(1)
    A4                             # map(4)
      01                           # unsigned(1)
      02                           # unsigned(2)
      20                           # negative(0)
      01                           # unsigned(1)
      21                           # negative(1)
      58 20                       # bytes(32)
        D7CC072DE2205BDC1537A543D53C60A6ACB62ECCD890C7FA27C9
        E354089BBE13
      22                           # negative(2)
      58 20                       # bytes(32)
        F95E1D4B851A2CC80FFF87D8E23F22AFB725D535E515D020731E
        79A3B4E47120
  18 3C                             # unsigned(60)
  41                                 # bytes(1)
    00
  18 3D                             # unsigned(61)
  44                                 # bytes(4)
    ABCD0000

```

Figure 5: Example CWT Claims Set with OSCORE parameters, CBOR encoded.

3.2.1. Salt Input Claim

The 'salt_input' claim provides a value that the Client requesting the Access Token wishes to use as a part of a salt with the RS, e.g., for deriving cryptographic material.

This parameter specifies the value of the salt input, encoded as a CBOR byte string.

3.2.2. Context ID Input Claim

The 'contextId_input' claim provides a value that the Client requesting the Access Token wishes to use with the RS, as a hint for a security context.

This parameter specifies the value of the Context ID input, encoded as a CBOR byte string.

4. Client-RS Communication

This section details the POST request and response to the /authz-info endpoint between the Client and the RS.

The proof-of-possession required to bind the Access Token to the Client is explicitly performed when the RS receives and verifies a request from the Client protected with Group OSCORE, either with the group mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]) or with the pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

In particular, the RS uses the Client's public key bound to the Access Token, either when verifying the signature of the request (if protected with the group mode), or when verifying the request as integrity-protected with pairwise keying material derived from the two peers' authentication credentials and asymmetric keys (if protected with the pairwise mode). In either case, the RS also authenticates the Client.

Similarly, when receiving a protected response from the RS, the Client uses the RS's public key either when verifying the signature of the response (if protected with the group mode), or when verifying the response as integrity-protected with pairwise keying material derived from the two peers' authentication credentials and asymmetric keys (if protected with the pairwise mode). In either case, the Client also authenticates the RS. Mutual authentication is only achieved after the client has successfully verified the protected response from the RS.

Therefore, an attacker using a stolen Access Token cannot generate a valid Group OSCORE message as protected through the Client's private key, and thus cannot prove possession of the pop-key bound to the Access Token. Also, if a Client legitimately owns an Access Token but has not joined the OSCORE group, it cannot generate a valid Group OSCORE message, as it does not store the necessary keying material shared among the group members.

Furthermore, a Client C1 is supposed to obtain a valid Access Token from the AS, as including the public key associated with its own private key used in the OSCORE group, together with its own Sender ID in that OSCORE group (see Section 3.1). This allows the RS receiving an Access Token to verify with the Group Manager of that OSCORE group whether such a Client has indeed that Sender ID and an authentication credential including that public key in the OSCORE group.

As a consequence, a different Client C2, also member of the same OSCORE group, is not able to impersonate C1, by: i) getting a valid Access Token, specifying the Sender ID of C1 and a different (made-up) public key; ii) successfully posting the Access Token to RS; and then iii) attempting to communicate using Group OSCORE impersonating C1, while blaming C1 for the consequences.

4.1. C-to-RS POST to authz-info Endpoint

The Client posts the Access Token to the /authz-info endpoint of the RS, as defined in Section 5.10.1 of [I-D.ietf-ace-oauth-authz].

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

The RS MUST verify the validity of the Access Token as defined in Section 5.10.1 of [I-D.ietf-ace-oauth-authz], with the following additions.

- * The RS MUST check that the claims 'salt_input', 'contextId_input' and 'cnf' are included in the Access Token.
- * The RS considers: the content of the 'contextId_input' claim as the GID of the OSCORE group; the content of the 'salt_input' claim as the Sender ID that the Client has in the group; and the content of the 'cnf' claim as the COSE Key equivalent to the authentication credential that the Client uses in the group.

The RS MUST check whether it already stores an authentication credential associated with the pair (GID, Sender ID) above, such that the COSE Key specified in the 'cnf' claim is its equivalent COSE Key.

If this is not the case, the RS MUST request the Client's authentication credential to the Group Manager of the OSCORE group as described in Section 10 of [I-D.ietf-ace-key-groupcomm-oscore], specifying the Client's Sender ID in the OSCORE group, i.e., the value of the 'salt_input' claim. Then, the RS performs the following actions.

- The RS MUST check whether the Client's authentication credential retrieved from the Group Manager is such that the COSE Key specified in the 'cnf' claim of the Access Token is its equivalent COSE Key.
- The RS MUST check that the Client's Sender ID provided by the Group Manager together with the Client's authentication credential matches the one retrieved from the 'salt_input' claim of the Access Token.

If any of the checks above fails, the RS MUST consider the Access Token non valid, and MUST respond to the Client with an error response code equivalent to the CoAP code 4.00 (Bad Request).

If the Access Token is valid and further checks on its content are successful, the RS associates the authorization information from the Access Token with the Group OSCORE Security Context.

In particular, the RS associates the authorization information from the Access Token with the 3-tuple (GID, SaltInput, AuthCred), where GID is the Group Identifier of the OSCORE Group, while SaltInput and AuthCred are the Sender ID and the authentication credential that the Client uses in that OSCORE group, respectively.

The RS MUST keep this association up-to-date over time, as the 3-tuple (GID, SaltInput, AuthCred) associated with the Access Token might change. In particular:

- * If the OSCORE group is rekeyed (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm] and Section 20 of [I-D.ietf-ace-key-groupcomm-oscore]), the Group Identifier also changes in the group, and the new one replaces the current 'GID' value in the 3-tuple.
- * If the Client requests and obtains a new OSCORE Sender ID from the Group Manager (see Section 2.5.3.1 of [I-D.ietf-core-oscore-groupcomm] and Section 9 of [I-D.ietf-ace-key-groupcomm-oscore]), the new Sender ID replaces the current 'SaltInput' value in the 3-tuple.

Finally, the RS MUST send a 2.01 (Created) response to the Client, as defined in Section 5.10.1 of [I-D.ietf-ace-oauth-authz].

4.3. Client-RS Secure Communication

When previously joining the OSCORE group, both the Client and RS have already established the related Group OSCORE Security Context to communicate as group members. Therefore, they can simply start to securely communicate using Group OSCORE, without deriving any additional keying material or security association.

4.3.1. Client Side

After having received the 2.01 (Created) response from the RS, following the POST request to the authz-info endpoint, the Client starts the communication with the RS, by sending a request protected with Group OSCORE using the Group OSCORE Security Context [I-D.ietf-core-oscore-groupcomm].

When communicating with the RS to access the resources as specified by the authorization information, the Client MUST use the Group OSCORE Security Context of the OSCORE group, whose GID was specified in the 'context_id' parameter of the Token request.

4.3.2. Resource Server Side

After successful validation of the Access Token as defined in Section 4.2 and after having sent the 2.01 (Created) response, the RS can start to communicate with the Client using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

When processing an incoming request protected with Group OSCORE, the RS MUST consider as valid Client's authentication credential only the one associated to the stored Access Token. As defined in Section 4.5, a possible change of authentication credential requires the Client to upload to the RS a new Access Token bound to the new authentication credential.

Additionally, for every incoming request, if Group OSCORE verification succeeds, the verification of access rights is performed as described in Section 4.4.

After the expiration of the Access Token related to a Group OSCORE Security Context, if the Client uses the Group OSCORE Security Context to send a request for any resource intended for OSCORE group members and that requires an active Access Token, the RS MUST respond with a 4.01 (Unauthorized) error message protected with the Group OSCORE Security Context.

4.4. Access Rights Verification

The RS MUST follow the procedures defined in Section 5.10.2 of [I-D.ietf-ace-oauth-authz]. If an RS receives a Group OSCORE-protected request from a Client, the RS processes it according to [I-D.ietf-core-oscore-groupcomm].

If the Group OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the authorization information from the Access Token associated with the Group OSCORE Security Context. Then, the RS MUST verify that the action requested on the resource is authorized.

The response code MUST be 4.01 (Unauthorized) if the RS has no valid Access Token for the Client. If the RS has an Access Token for the Client but no actions are authorized on the target resource, the RS MUST reject the request with a 4.03 (Forbidden). If the RS has an Access Token for the Client but the requested action is not authorized, the RS MUST reject the request with a 4.05 (Method Not Allowed).

4.5. Change of Client's Authentication Credential in the Group

During its membership in the OSCORE group, the client might change the authentication credential it uses in the group. When this happens, the Client uploads the new authentication credential to the Group Manager, as defined in Section 11 of [I-D.ietf-ace-key-groupcomm-oscore].

After that, and in order to continue communicating with the RS, the Client MUST perform the following actions.

1. The Client requests a new Access Token to the AS, as defined in Section 3. In particular, when sending the POST request as defined in Section 3.1, the Client indicates:
 - * The current Group Identifier of the OSCORE group, as value of the 'context_id' parameter.
 - * The current Sender ID it has in the OSCORE group, as value of the 'salt_input' parameter.
 - * The public key of the new authentication credential it uses in the OSCORE group, as value of the 'req_cnf' parameter. In particular, the specified public key is the COSE Key equivalent to the new authentication credential that the Client uses in the OSCORE group.

- * The proof-of-possession (PoP) evidence corresponding to the public key of the new authentication credential, as value of the 'client_cred_verify' or 'client_cred_verify_mac' parameter.
2. After receiving the response from the AS (see Section 3.2), the Client performs the same exchanges with the RS as defined in Section 4.

When receiving the new Access Token, the RS performs the same steps defined in Section 4.2, with the following addition, in case the new Access Token is successfully verified and stored. The RS also deletes the old Access Token, i.e., the one whose associated 3-tuple has the same GID and SaltInput values as in the 3-tuple including the new authentication credential of the Client and associated with the new Access Token.

5. Secure Communication with the AS

As specified in the ACE framework (see Sections 5.8 and 5.9 of [I-D.ietf-ace-oauth-authz]), the requesting entity (RS and/or Client) and the AS communicate via the /token or /introspection endpoint. The use of CoAP and OSCORE [RFC8613] for this communication is RECOMMENDED in this profile. Other protocols fulfilling the security requirements defined in Sections 5 and 6 of [I-D.ietf-ace-oauth-authz] (such as HTTP and DTLS or TLS) MAY be used instead.

If OSCORE [RFC8613] is used, the requesting entity and the AS are expected to have a pre-established Security Context in place. How this Security Context is established is out of the scope of this profile. Furthermore, the requesting entity and the AS communicate using OSCORE through the /introspection endpoint as specified in Section 5.9 of [I-D.ietf-ace-oauth-authz], and through the /token endpoint as specified in Section 5.8 of [I-D.ietf-ace-oauth-authz].

6. Discarding the Security Context

As members of an OSCORE group, the Client and the RS may independently leave the group or be forced to, e.g., if compromised or suspected so. Upon leaving the OSCORE group, the Client or RS also discards the Group OSCORE Security Context, which may anyway be renewed by the Group Manager through a group rekeying process (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]).

The Client or RS can acquire a new Group OSCORE Security Context, by re-joining the OSCORE group, e.g., by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. In such a case, the Client SHOULD request a new Access Token and post it to the RS.

7. CBOR Mappings

The new parameters defined in this document MUST be mapped to CBOR types as specified in Figure 6, using the given integer abbreviation for the map key.

Parameter name	CBOR Key	Value Type
context_id	TBD	bstr
salt_input	TBD	bstr
client_cred_verify	TBD	bstr
client_cred_verify_mac	TBD	bstr

Figure 6: CBOR mappings for new parameters.

The new claims defined in this document MUST be mapped to CBOR types as specified in Figure 7, using the given integer abbreviation for the map key.

Claim name	CBOR Key	Value Type
salt_input	TBD	bstr
contextId_input	TBD	bstr

Figure 7: CBOR mappings for new claims.

8. Security Considerations

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

The proof-of-possession (PoP) key bound to an Access Token is always an asymmetric key, i.e., the public key that the Client uses in the OSCORE group. This means that there is never a same shared secret used as PoP key with possible multiple RSs. Therefore, it is possible and safe for the AS to issue an Access Token whose audience comprises multiple RSs.

In such a case, as per Section 6.1 of [I-D.ietf-ace-oauth-authz], the AS has to ensure the integrity protection of the Access Token by protecting it through an asymmetric signature. In addition, the used audience has to correctly identify all the RSs that are intended recipients of the Access Token. As a particular case, the audience can be the name of the OSCORE group, if the Access Token is intended to all the RSs in that group.

Furthermore, this document inherits the general security considerations about Group OSCORE [I-D.ietf-core-oscore-groupcomm], as to the specific use of Group OSCORE according to this profile.

Group OSCORE is designed to secure point-to-point as well as point-to-multipoint communications, providing a secure binding between a single request and multiple corresponding responses. In particular, Group OSCORE fulfills the same security requirements of OSCORE, for group requests and responses.

Group OSCORE ensures source authentication of messages both in group mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]) and in pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

When protecting an outgoing message in group mode, the sender uses its private key to compute a digital signature, which is embedded in the protected message. The group mode can be used to protect messages sent over multicast to multiple recipients, or sent over unicast to one recipient.

When protecting an outgoing message in pairwise mode, the sender uses a pairwise symmetric key, as derived from the asymmetric keys of the two peers exchanging the message. The pairwise mode can be used to protect only messages intended to one recipient.

9. Privacy Considerations

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

As this profile uses Group OSCORE, the privacy considerations from [I-D.ietf-core-oscore-groupcomm] apply to this document as well.

An unprotected response to an unauthorized request may disclose information about the RS and/or its existing relationship with the Client. It is advisable to include as little information as possible in an unencrypted response. However, since both the Client and the RS share a Group OSCORE Security Context, unauthorized, yet protected requests are followed by protected responses, which can thus include more detailed information.

Although it may be encrypted, the Access Token is sent in the clear to the /authz-info endpoint at the RS. Thus, if the Client uses the same single Access Token from multiple locations with multiple Resource Servers, it can risk being tracked through the Access Token's value.

Note that, even though communications are protected with Group OSCORE, some information might still leak, due to the observable size, source address and destination address of exchanged messages.

10. IANA Considerations

This document has the following actions for IANA.

10.1. ACE Profile Registry

IANA is asked to add the following entry to the "ACE Profile" registry defined in Section 8.8 of [I-D.ietf-ace-oauth-authz].

- * Name: coap_group_oscore
- * Description: Profile to secure communications between constrained nodes using the Authentication and Authorization for Constrained Environments framework, by enabling authentication and fine-grained authorization of members of an OSCORE group, that use a pre-established Group OSCORE Security Context to communicate with Group OSCORE. Optionally, the dual mode defined in Appendix A additionally establishes a pairwise OSCORE Security Context, and thus also enables OSCORE communication between two members of the OSCORE group.
- * CBOR Value: TBD (value between 1 and 255)
- * Reference: [[this document]]

10.2. OAuth Parameters Registry

IANA is asked to add the following entries to the "OAuth Parameters" registry.

- * Name: "context_id"
- * Parameter Usage Location: token request
- * Change Controller: IESG
- * Specification Document(s): Section 3.1.1 of [[this document]]

- * Name: "salt_input"
- * Parameter Usage Location: token request
- * Change Controller: IESG
- * Specification Document(s): Section 3.1.2 of [[this document]]

- * Name: "client_cred_verify"
- * Parameter Usage Location: token request
- * Change Controller: IESG
- * Specification Document(s): Section 3.1.3 of [[this document]]

- * Name: "client_cred_verify_mac"
- * Parameter Usage Location: token request
- * Change Controller: IESG
- * Specification Document(s): Section 3.1.4 of [[this document]]

- * Name: "client_cred"
- * Parameter Usage Location: token request
- * Change Controller: IESG
- * Specification Document(s): Appendix A.2.1.1 of [[this document]]

10.3. OAuth Parameters CBOR Mappings Registry

IANA is asked to add the following entries to the "OAuth Parameters CBOR Mappings" registry defined in Section 8.10 of [I-D.ietf-ace-oauth-authz].

- * Name: "context_id"
- * CBOR Key: TBD
- * Value Type: bstr
- * Reference: Section 3.1.1 of [[this document]]

- * Name: "salt_input"
- * CBOR Key: TBD
- * Value Type: bstr
- * Reference: Section 3.1.2 of [[this document]]

- * Name: "client_cred_verify"
- * CBOR Key: TBD
- * Value Type: bstr
- * Reference: Section 3.1.3 of [[this document]]

- * Name: "client_cred_verify_mac"
- * CBOR Key: TBD
- * Value Type: bstr
- * Reference: Section 3.1.4 of [[this document]]

- * Name: "client_cred"
- * CBOR Key: TBD
- * Value Type: bstr

- * Reference: Appendix A.2.1.1 of [[this document]]

10.4. CBOR Web Token Claims Registry

IANA is asked to add the following entries to the "CBOR Web Token Claims" registry.

- * Claim Name: "salt_input"
- * Claim Description: Client provided salt input
- * JWT Claim Name: "N/A"
- * Claim Key: TBD
- * Claim Value Type(s): bstr
- * Change Controller: IESG
- * Specification Document(s): Section 3.2.1 of [[this document]]

- * Claim Name: "contextId_input"
- * Claim Description: Client context id input
- * JWT Claim Name: "N/A"
- * Claim Key: TBD
- * Claim Value Type(s): bstr
- * Change Controller: IESG
- * Specification Document(s): Section 3.2.2 of [[this document]]

- * Claim Name: "client_cred"
- * Claim Description: Client Credential
- * JWT Claim Name: "N/A"
- * Claim Key: TBD
- * Claim Value Type(s): map
- * Change Controller: IESG

- * Specification Document(s): Appendix A.2.2.2 of [[this document]]

10.5. TLS Exporter Label Registry

IANA is asked to add the following entry to the "TLS Exporter Label" registry defined in Section 6 of [RFC5705] and updated in Section 12 of [RFC8447].

- * Value: EXPORTER-ACE-Sign-Challenge-Client-AS
- * DTLS-OK: Y
- * Recommended: N
- * Reference: [[this document]] (Section 3.1)

11. References

11.1. Normative References

- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.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-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.
- [I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-16, 7 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-16.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-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.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-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-14, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt>>.

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

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

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

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

[NIST-800-56A]

Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018, <<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <<https://www.rfc-editor.org/info/rfc8447>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

11.2. Informative References

- [I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.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-15, 1 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-mqtt-tls-profile-15.txt>>.
- [I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>>.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in

Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022,
<<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>>.

- [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>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Dual Mode (Group OSCORE & OSCORE)

This appendix defines the dual mode of this profile, which allows using both OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm] as security protocols, by still relying on a single Access Token.

That is, the dual mode of this profile specifies how a Client uses CoAP [RFC7252] to communicate to a single Resource Server, or CoAP over IP multicast [I-D.ietf-core-groupcomm-bis] to communicate to multiple Resource Servers that are members of a group and share a common set of resources.

In particular, the dual mode of this profile uses two complementary security protocols to provide secure communication between the Client and the Resource Server(s). That is, it defines the use of either OSCORE or Group OSCORE to protect unicast requests addressed to a single Resource Server, as well as possible responses. Additionally, it defines the use of Group OSCORE to protect multicast requests sent to a group of Resource Servers, as well as possible individual responses. Like in the main mode of this profile, the Client and the Resource Servers need to have already joined the same OSCORE group, for instance by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore], which is also based on ACE.

The Client proves its access to be authorized to the Resource Server by using an Access Token, which is bound to a key (the proof-of-possession key). This profile mode uses OSCORE to achieve proof of possession, and OSCORE or Group OSCORE to achieve server authentication.

Unlike in the main mode of this profile, where a public key is used as pop-key, this dual mode uses OSCORE-related, symmetric keying material as pop-key instead. Furthermore, this dual mode provides proof of Client's membership to the correct OSCORE group, by securely binding the pre-established Group OSCORE Security Context to the pairwise OSCORE Security Context newly established between the Client and the Resource Server.

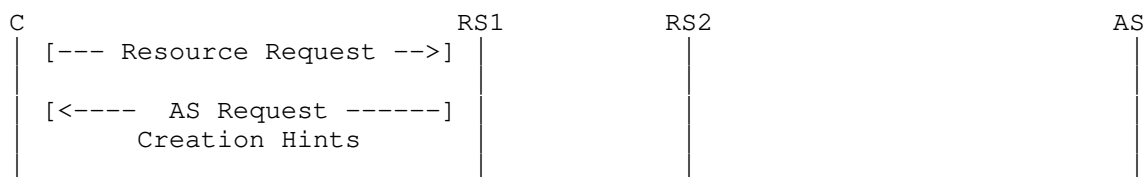
In addition to the terminology used for the main mode of this profile, the rest of this appendix refers also to "pairwise OSCORE Security Context" as to an OSCORE Security Context established between only one Client and one Resource Server, and used to communicate with OSCORE [RFC8613].

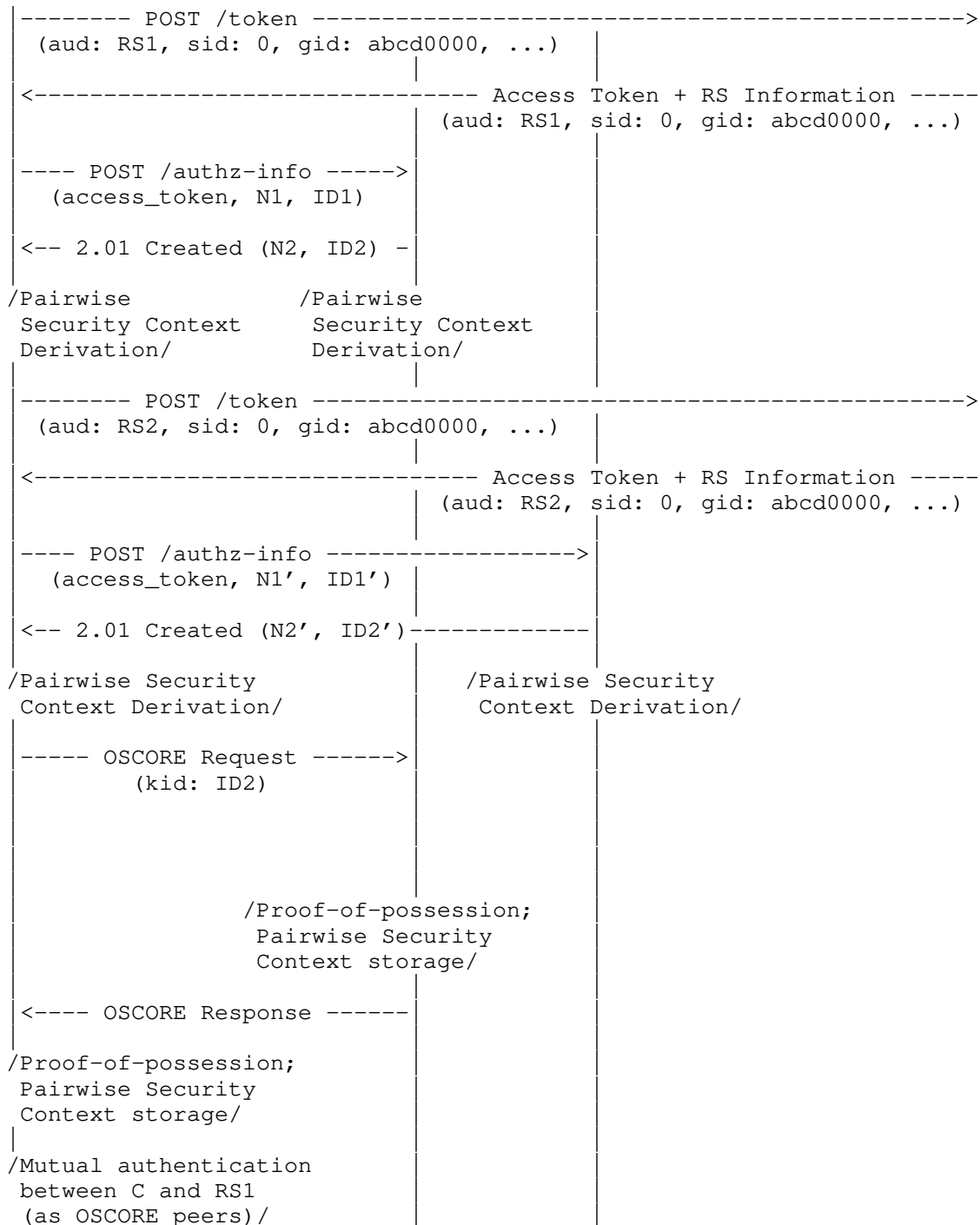
A.1. Protocol Overview

This section provides an overview on how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to secure communications between a Client and a (set of) Resource Server(s) using OSCORE [RFC8613] and/or Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Just as for main mode of this profile overviewed in Section 2, the process for joining the OSCORE group through the respective Group Manager as defined in [I-D.ietf-ace-key-groupcomm-oscore] must take place before the process described in the rest of this section, and is out of the scope of this profile.

An overview of the protocol flow for the dual mode of this profile is shown in Figure 8. In the figure, it is assumed that both RS1 and RS2 are associated with the same AS. It is also assumed that C, RS1 and RS2 have previously joined an OSCORE group with Group Identifier (gid) "abcd0000", and got assigned Sender ID (sid) "0", "1" and "2" in the group, respectively. The names of messages coincide with those of [I-D.ietf-ace-oauth-authz] when applicable.





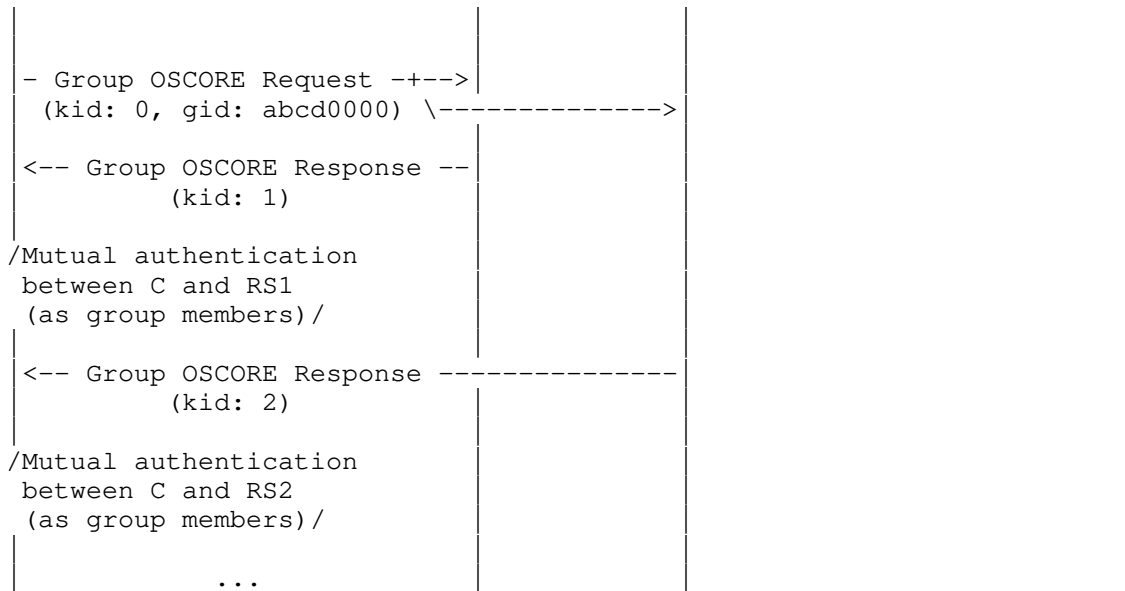


Figure 8: Protocol Overview.

A.1.1. Pre-Conditions

The same pre-conditions for the main mode of this profile (see Section 2.1) hold for the dual mode described in this appendix.

A.1.2. Access Token Posting

After having retrieved the Access Token from the AS, the Client generates a nonce $N1$ and an identifier $ID1$ unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts. The client then posts both the Access Token, $N1$ and its chosen ID to the RS, using the `/authz-info` endpoint and mechanisms specified in Section 5.10 of [I-D.ietf-ace-oauth-authz] and `Content-Format = application/ace+cbor`.

When using the dual mode of this profile, the communication with the `authz-info` endpoint is not protected, except for update of access rights. Note that, when using the dual mode, this request can alternatively be protected with Group OSCORE, using the Group OSCORE Security Context paired with the pairwise OSCORE Security Context originally established with the first Access Token posting.

If the Access Token is valid, the RS replies to this POST request with a 2.01 (Created) response with Content-Format = application/ace+cbor, which in a CBOR map contains a nonce N2 and an identifier ID2 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts.

A.1.3. Setup of the Pairwise OSCORE Security Context

After sending the 2.01 (Created) response, the RS sets the ID Context of the pairwise OSCORE Security Context (see Section 3 of [RFC8613]) to the Group Identifier of the OSCORE group specified in the Access Token, concatenated with N1, concatenated with N2, concatenated with the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' claim of the Access Token.

Then, the RS derives the complete pairwise OSCORE Security Context associated with the received Access Token, following Section 3.2 of [RFC8613]. In practice, the RS maintains a collection of Security Contexts with associated authorization information, for all the clients that it is currently communicating with, and the authorization information is a policy used as input when processing requests from those clients.

During the derivation process, the RS uses: the ID Context above; the exchanged nonces N1 and N2; the identifier ID1 received from the Client, set as its own OSCORE Sender ID; the identifier ID2 provided to the Client, set as its Recipient ID for the Client; and the parameters in the Access Token. The derivation process uses also the Master Secret of the OSCORE group, that the RS knows as a group member, as well as the Sender ID of the Client in the OSCORE group, which is specified in the Access Token. This ensures that the pairwise OSCORE Security Context is securely bound to the Group OSCORE Security Context of the OSCORE group.

Finally, the RS stores the association between i) the authorization information from the Access Token; and ii) the Group Identifier of the OSCORE group together with the Sender ID and the authentication credential of the Client in that group.

After having received the nonce N2, the Client sets the ID Context in its pairwise OSCORE Security Context (see Section 3 of [RFC8613]) to the Group Identifier of the OSCORE group, concatenated with N1, concatenated with N2, concatenated with the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' parameter of the Access Token response from the AS. Then, the Client derives the complete pairwise OSCORE Security Context, following Section 3.2 of [RFC8613].

During the derivation process, the Client uses: the ID Context above, the exchanged nonces N1 and N2; the identifier ID1 provided to the RS, set as its own Recipient ID for the RS; the identifier ID2 received from the RS, set as its own OSCORE Sender ID; and the parameters received from the AS. The derivation process uses also the Master Secret of the OSCORE group, that the Client knows as a group member, as well as its own Sender ID in the OSCORE group.

When the Client communicates with the RS using the pairwise OSCORE Security Context, the RS achieves proof-of-possession of the credentials bound to the Access Token. Also, the RS verifies that the Client is a legitimate member of the OSCORE group.

A.1.4. Secure Communication

Other than starting the communication with the RS using Group OSCORE as described in Section 4.3, the Client can send to the RS a request protected with OSCORE, using the pairwise OSCORE Security Context.

If the request is successfully verified, then the RS stores the pairwise OSCORE Security Context, and uses it to protect the possible response, as well as further communications with the Client, until the Access Token is deleted, due to, for example, expiration. This pairwise OSCORE Security Context is discarded when an Access Token (whether the same or different) is used to successfully derive a new pairwise OSCORE Security Context.

As discussed in Section 7 of [I-D.ietf-ace-oscore-profile], the use of random nonces N1 and N2 during the exchange between the Client and the RS prevents the reuse of an Authenticated Encryption with Associated Data (AEAD) nonce/key pair for two different messages. Reuse might otherwise occur when Client and RS derive a new pairwise OSCORE Security Context from an existing (non-expired) Access Token, e.g., in case of reboot of either the Client or the RS, and might lead to loss of both confidentiality and integrity.

Additionally, just as per the main mode of this profile (see Section 4.3), the Client and RS can also securely communicate by protecting messages with Group OSCORE, using the Group OSCORE Security Context already established upon joining the OSCORE group.

A.2. Client-AS Communication

This section details the Access Token POST Request that the Client sends to the /token endpoint of the AS, as well as the related Access Token response.

Section 3.2 of [RFC8613] defines how to derive a pairwise OSCORE Security Context based on a shared Master Secret and a set of other parameters, established between the OSCORE client and server, which the client receives from the AS in this exchange.

The proof-of-possession key (pop-key) received from the AS in this exchange MUST be used to build the Master Secret in OSCORE (see Appendix A.3.3 and Appendix A.3.4).

A.2.1. C-to-AS: POST to Token Endpoint

The Client-to-AS request is specified in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. The Client MUST send this POST request to the /token endpoint over a secure channel that guarantees authentication, message integrity and confidentiality.

The POST request is formatted as the analogous Client-to-AS request in the main mode of this profile (see Section 3.1), with the following modifications.

- * The parameter 'req_cnf' MUST NOT be included in the payload.
- * The parameter 'client_cred', defined in Appendix A.2.1.1 of this document, MUST be included in the payload. This parameter specifies the public key that the Client uses in the OSCORE group, whose identifier is indicated in the 'context_id' parameter. In particular, the specified public key is the COSE Key equivalent to the authentication credential that the Client uses in the OSCORE group.
- * The proof-of-possession (PoP) evidence included in the 'client_cred_verify' or 'client_cred_verify_mac' parameter is computed by using the Client's private key associated with the public key in the 'client_cred' parameter above.

An example of such a request is shown in Figure 9.


```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "audience" : "tempSensor4711",
  "scope" : "read",
  "context_id" : h'abcd0000',
  "salt_input" : h'00',
  "client_cred" : {
    "COSE_Key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
        27c9e354089bbe13',
      "y" : h'f95eld4b851a2cc80fff87d8e23f22afb725d535e515d020
        731e79a3b4e47120'
    }
  },
  "client_cred_verify" : h'...'
  (signature content omitted for brevity),
}
```

Figure 9: Example C-to-AS POST /token request for an Access Token bound to a symmetric key.

Later on, the Client may want to update its current access rights, without changing the existing pairwise OSCORE Security Context with the RS. In this case, the Client MUST include in its POST request to the /token endpoint a 'req_cnf' parameter, defined in Section 3.1 of [I-D.ietf-ace-oauth-params], which MUST include a 'kid' field, as defined in Section 3.1 of [RFC8747]. The 'kid' field has as value a CBOR byte string containing the OSCORE_Input_Material Identifier (assigned as discussed in Appendix A.2.2).

This identifier, together with other information such as audience, can be used by the AS to determine the shared secret bound to the proof-of-possession Access Token and therefore MUST identify a symmetric key that was previously generated by the AS as a shared secret for the communication between the Client and the RS. The AS MUST verify that the received value identifies a proof-of-possession key that has previously been issued to the requesting Client. If that is not the case, the Client-to-AS request MUST be declined with the error code "invalid_request" as defined in Section 5.8.3 of [I-D.ietf-ace-oauth-authz].

This POST request for updating the access rights of an Access Token SHOULD NOT include the parameters 'salt_input', 'context_id', 'client_cred' and 'client_cred_verify'. An exception is the case defined in Appendix A.3.6, where the Client, following a change of authentication credential in the OSCORE group, requests a new Access Token associated with the public key of the new authentication credential, while still without changing the existing pairwise OSCORE Security Context with the RS.

An example of such a request is shown in Figure 10.

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

Figure 10: Example C-to-AS POST /token request for updating rights to an Access Token bound to a symmetric key.

A.2.1.1. 'client_cred' Parameter

The 'client_cred' parameter is an OPTIONAL parameter of the Access Token request message defined in Section 5.8.1. of [I-D.ietf-ace-oauth-authz]. This parameter provides an asymmetric key that the Client wishes to use as its own public key, but which is not used as proof-of-possession key.

This parameter follows the syntax of the 'cnf' claim from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. Alternative Value Types defined in future specifications are fine to consider if indicating a non-encrypted asymmetric key.

A.2.2. AS-to-C: Access Token

After having verified the POST request to the /token endpoint and that the Client is authorized to obtain an Access Token corresponding to its Access Token request, the AS MUST verify the proof-of-possession (PoP) evidence.

In particular, the AS proceeds as defined in Section 3.2, with the difference that it uses the public key specified in the 'client_cred' parameter as public key of the Client.

If both the 'client_cred_verify' and 'client_cred_verify_mac' parameters are present, or if the verification of the PoP evidence fails, the AS considers the Client request invalid. The AS does not perform this operation when asked to update a previously released Access Token.

If all verifications are successful, the AS responds as defined in Section 5.8.2 of [I-D.ietf-ace-oauth-authz]. If the Client request was invalid, or not authorized, the AS returns an error response as described in Section 5.8.3 of [I-D.ietf-ace-oauth-authz].

The AS can signal that the use of OSCORE and Group OSCORE is REQUIRED for a specific Access Token by including the "ace_profile" parameter with the value "coap_group_oscore" in the Access Token response. This means that the Client MUST use OSCORE and/or Group OSCORE towards all the Resource Servers for which this Access Token is valid.

In particular, the Client MUST follow Appendix A.3.3 to derive the pairwise OSCORE Security Context to use for communications with the RS. Instead, the Client has already established the related Group OSCORE Security Context to communicate with members of the OSCORE group, upon previously joining that group.

Usually, it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile signaling can be omitted.

In contrast with the main mode of this profile, the Access Token response to the Client is analogous to the one in the OSCORE profile of ACE, as described in Section 3.2 of [I-D.ietf-ace-oscore-profile]. In particular, the AS provides an OSCORE_Input_Material object, which is defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile] and included in the 'cnf' parameter (see Section 3.2 of [I-D.ietf-ace-oauth-params]) of the Access Token response.

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

In the issued Access Token, the AS MUST include as metadata the same information as defined in the main mode of this profile (see Section 3.2) with the following modifications.

- * The public key that the client uses in the OSCORE group and specified in the 'client_cred' parameter of the Token request (see Appendix A.2.1) MUST also be included in the Access Token.

If the Access Token is a CWT, the AS MUST include it in the 'client_cred' claim of the Access Token, defined in Appendix A.2.2.2 of this document. In particular, the 'client_cred' claim includes the same COSE Key specified in the 'client_cred' parameter of the Token Request, i.e., the COSE Key equivalent to the authentication credential that the Client uses in the OSCORE group.

- * The OSCORE_Input_Material specified in the 'cnf' parameter of the Access Token response MUST also be included in the Access Token. If the Access Token is a CWT, the same OSCORE_Input_Material included in the 'cnf' parameter of the Access Token response MUST be included in the 'osc' field of the 'cnf' claim of the Access Token (see Section 3.2 of [I-D.ietf-ace-oscore-profile]).

Figure 11 shows an example of such an AS response. The access token has been truncated for readability.

```
Header: Created (Code=2.01)
Content-Type: "application/ace+cbor"
Payload:
{
  "access_token" : h'8343a1010aa2044c53 ...'
    (remainder of CWT omitted for brevity),
  "ace_profile" : "coap_group_oscore",
  "expires_in" : 3600,
  "cnf" : {
    "osc" : {
      "alg" : AES-CCM-16-64-128,
      "id" : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f',
      "salt" : h'1122',
      "contextId" : h'99'
    }
  }
}
```

Figure 11: Example AS-to-C Access Token response with the Group OSCORE profile.

Figure 12 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : 1360189224,
  "exp" : 1360289224,
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "osc" : {
      "alg" : AES-CCM-16-64-128,
      "id" : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f',
      "salt" : h'1122',
      "contextId" : h'99'
    },
    "salt_input" : h'00',
    "contextId_input" : h'abcd0000',
    "client_cred" : {
      "COSE_Key" : {
        "kty" : EC2,
        "crv" : P-256,
        "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
          27c9e354089bbe13',
        "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
          731e79a3b4e47120'
      }
    }
  }
}

```

Figure 12: Example CWT with OSCORE parameters.

The same CWT as in Figure 12 and encoded in CBOR is shown in Figure 13, using the value abbreviations defined in [I-D.ietf-ace-oauth-authz] and [RFC8747].

NOTE: it should be checked (and in case fixed) that the values used below (which are not yet registered) are the final values registered in IANA.

```

A8                                     # map(8)
  03                                   # unsigned(3)
  76                                   # text(22)
    74656D7053656E736F72496E4C6976696E67526F6F6D
  06                                   # unsigned(6)
  1A 5112D728                         # unsigned(1360189224)
  04                                   # unsigned(4)
  1A 51145DC8                         # unsigned(1360289224)
  09                                   # unsigned(9)
  78 18                               # text(24)
    74656D70657261747572655F67206669726D776172655F70

```

```

08                                     # unsigned(8)
A1                                     # map(1)
  04                                     # unsigned(4)
  A5                                     # map(5)
    04                                     # unsigned(4)
    0A                                     # unsigned(10)
    00                                     # unsigned(0)
    41                                     # bytes(1)
      01                                     # "\x01"
    02                                     # unsigned(2)
    50                                     # bytes(16)
      F9AF838368E353E78888E1426BD94E6F
    05                                     # unsigned(5)
    42                                     # bytes(2)
      1122                                     # "\x11\""
    06                                     # unsigned(6)
    41                                     # bytes(1)
      99                                     # "\x99"
18 3C                                     # unsigned(60)
41                                     # bytes(1)
  00
18 3D                                     # unsigned(61)
44                                     # bytes(4)
  ABCD0000
18 3E                                     # unsigned(62)
A1                                     # map(1)
  01                                     # unsigned(1)
  A4                                     # map(4)
    01                                     # unsigned(1)
    02                                     # unsigned(2)
    20                                     # negative(0)
    01                                     # unsigned(1)
    21                                     # negative(1)
    58 20                                     # bytes(32)
      D7CC072DE2205BDC1537A543D53C60A6ACB62ECCD890C7FA27C9
      E354089BBE13
    22                                     # negative(2)
    58 20                                     # bytes(32)
      F95E1D4B851A2CC80FFF87D8E23F22AFB725D535E515D020731E
      79A3B4E47120

```

Figure 13: Example CWT with OSCORE parameters.

If the Client has requested an update to its access rights using the same pairwise OSCORE Security Context, which is valid and authorized, the AS MUST omit the 'cnf' parameter in the response to the client.

Instead, the updated Access Token conveyed in the AS-to-C response MUST include a 'cnf' claim specifying a 'kid' field, as defined in Section 3.1 of [RFC8747]. The response from the AS MUST carry the OSCORE Input Material identifier in the 'kid' field within the 'cnf' claim of the Access Token. That is, the 'kid' field is a CBOR byte string, with value the same value of the 'kid' field of the 'req_cnf' parameter from the C-to-AS request for updating rights to the Access Token (see Figure 10). This information needs to be included in the Access Token, in order for the RS to identify the previously generated pairwise OSCORE Security Context.

Figure 14 shows an example of such an AS response. The Access Token has been truncated for readability.

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

Figure 14: Example AS-to-C Access Token response with the Group OSCORE profile, for update of access rights.

Figure 15 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim for update of access rights.

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

Figure 15: Example CWT with OSCORE parameters for update of access rights.

A.2.2.1. Public Key Hash as Client Credential

As a possible optimization to limit the size of the Access Token, the AS may specify as value of the 'client_cred' claim simply the hash of the Client's public key, i.e., the hash of the COSE Key K equivalent to the authentication credential that the Client uses in the OSCORE group.

The specifically used hash-function MUST be collision-resistant on byte-strings, and MUST be selected from the "Named Information Hash Algorithm" Registry defined in Section 9.4 of [RFC6920].

In particular, the AS provides the Client with an Access Token as defined in Appendix A.2.2, with the following differences.

The AS prepares INPUT_HASH as follows, with | denoting byte string concatenation.

- * If the equivalent COSE Key K has COSE Key Type OKP, INPUT_HASH = i, where 'i' is the x-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, encoded as a CBOR byte string.
- * If the equivalent COSE Key K has COSE Key Type EC2, INPUT_HASH = (i_1 | i_2), where 'i_1' and 'i_2' are the x-parameter and y-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, respectively, each encoded as a CBOR byte string.
- * If the equivalent COSE Key K has COSE Key Type RSA, INPUT_HASH = (i_1 | i_2), where 'i_1' and 'i_2' are the n-parameter and e-parameter of the COSE_Key specified in the 'client_cred' parameter of the Token request, respectively, each encoded as a CBOR byte string.

Then, the AS hashes INPUT_HASH according to the procedure described in [RFC6920], with the output OUTPUT_HASH in binary format, as described in Section 6 of [RFC6920].

Finally, the AS includes a single entry within the 'client_cred' claim of the Access Token. This entry has label "kid" (3) defined in Section 3.1 of [RFC8747], and value a CBOR byte string wrapping OUTPUT_HASH.

Upon receiving the Access Token, the RS processes it according to Appendix A.3.2, with the following differences.

The RS considers: the content of the 'contextId_input' claim as the GID of the OSCORE group; the content of the 'salt_input' claim as the Sender ID that the Client has in the group; and the content of the 'client_cred' claim as the hash RECEIVED_HASH of a COSE Key equivalent to the authentication credential that the Client uses in the group.

The RS MUST check whether it already stores an authentication credential associated with the pair (GID, Sender ID) above, such that the recomputed hash NEW_HASH of its equivalent COSE Key matches with RECEIVED_HASH from the 'client_cred' claim.

If this is not the case, the RS MUST request the Client's authentication credential to the Group Manager of the OSCORE group as described in Section 10 of [I-D.ietf-ace-key-groupcomm-oscure], specifying the Client's Sender ID in the OSCORE group, i.e., the value of the 'salt_input' claim. Then, the RS performs the following actions.

- * The RS MUST check whether RECEIVED_HASH matches with the recomputed hash NEW_HASH of a COSE Key equivalent to the Client's authentication credential retrieved from the Group Manager.
- * The RS MUST check that the Client's Sender ID provided by the Group Manager together with the Client's authentication credential matches the one retrieved from the 'salt_input' claim of the Access Token.

The RS MUST calculate NEW_HASH using the same method used by the AS described above, and using the same hash function. The hash function to use can be determined from the information conveyed in the 'client_cred' claim, as the procedure described in [RFC6920] also encodes the used hash function as metadata of the hash value.

A.2.2.2. Client Credential Claim

The 'client_cred' claim provides an asymmetric key that the Client owning the Access Token wishes to use as its own public key, but which is not used as proof-of-possession key.

This parameter follows the syntax of the 'cnf' claim from Section 3.1 of [RFC8747] when including Value Type "COSE_Key" (1) and specifying an asymmetric key. Alternative Value Types defined in future specifications are fine to consider, if indicating a non-encrypted asymmetric key or full-fledged authentication credential.

A.3. Client-RS Communication

This section details the POST request and response to the /authz-info endpoint between the Client and the RS. With respect to the exchanged messages and their content, the Client and the RS perform as defined in the OSCORE profile of ACE (see Section 4 of [I-D.ietf-ace-oscore-profile]).

That is, the Client generates a nonce N1 and posts it to the RS, together with: an identifier ID1 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts; and the Access Token that includes the material provisioned by the AS.

Then, the RS generates a nonce N2, and an identifier ID2 unique in the sets of its own Recipient IDs from its pairwise OSCORE Security Contexts. After that, the RS derives a pairwise OSCORE Security Context as described in Section 3.2 of [RFC8613]. In particular, it uses the two exchanged nonces and the two identifiers established with the Client, as well as two shared secrets together with additional pieces of information specified in the Access Token.

Both the client and the RS generate the pairwise OSCORE Security Context using the pop-key as part of the OSCORE Master Secret. In addition, the derivation of the pairwise OSCORE Security Context takes as input also information related to the OSCORE group, i.e., the Master Secret and Group Identifier of the group, as well as the Sender ID of the Client in the group. Hence, the derived pairwise OSCORE Security Context is also securely bound to the Group OSCORE Security Context of the OSCORE Group. Thus, the proof-of-possession required to bind the Access Token to the Client occurs after the first OSCORE message exchange.

Therefore, an attacker using a stolen Access Token cannot generate a valid pairwise OSCORE Security Context and thus cannot prove possession of the pop-key. Also, if a Client legitimately owns an Access Token but has not joined the OSCORE group, that Client cannot generate a valid pairwise OSCORE Security Context either, since it lacks the Master Secret used in the OSCORE group.

Besides, just as in the main mode (see Section 4), the RS is able to verify whether the Client has indeed the claimed Sender ID and authentication credential in the OSCORE group.

A.3.1. C-to-RS POST to authz-info Endpoint

The Client MUST generate a nonce N1, an OSCORE Recipient ID (ID1), and post them to the /authz-info endpoint of the RS together with the Access Token, as defined in the OSCORE profile of ACE (see Section 4.1 of [I-D.ietf-ace-oscore-profile]).

The same recommendations, considerations and behaviors defined in Section 4.1 of [I-D.ietf-ace-oscore-profile] hold.

If the Client has already posted a valid Access Token, has already established a pairwise OSCORE Security Context with the RS, and wants to update its access rights, the Client can do so by posting the new Access Token (retrieved from the AS and specifying the updated set of access rights) to the /authz-info endpoint.

The Client MUST protect the request using either the pairwise OSCORE Security Context established during the first Access Token exchange, or the Group OSCORE Security Context associated with that pairwise OSCORE Security Context.

In either case, the Client MUST only send the Access Token in the payload, i.e., no nonce or identifier are sent. After proper verification (see Section 4.2 of [I-D.ietf-ace-oscore-profile]), the new Access Token will supersede the old one at the RS, by replacing the corresponding authorization information. At the same time, the RS will maintain the same pairwise OSCORE Security Context and Group OSCORE Security Context, as now both associated with the new Access Token.

A.3.2. RS-to-C: 2.01 (Created)

The RS MUST verify the validity of the Access Token as defined in Section 4.2, with the following modifications.

- * If the POST request to /authz-info is not protected, the RS checks that the 'cnf' claim is included in the Access Token and that it contains an OSCORE_Input_Material object. Also, the RS checks that the 'salt_input', 'client_cred' and 'contextId_input' claims are included in the Access Token.
- * If the POST request to /authz-info is protected with the pairwise OSCORE Security Context shared with the Client or with the Group OSCORE Security Context of the OSCORE group, i.e., the Client is requesting an update of access rights, the RS checks that the 'cnf' claim is included in the Access Token and that it contains only the 'kid' field.

- * If the 'salt_input', 'client_cred' and 'contextId_input' claims are included in the Access Token, the RS extracts the content of 'client_cred'. Then, the RS considers that content as the COSE Key equivalent to the authentication credential that the Client uses in the group, whose GID is specified in the 'contextId_input' claim. The RS can compare this public key with the actual COSE Key equivalent to the authentication credential of the claimed Client, retrieved from its local storage or from the Group Manager (see Section 4.2).

If any of the checks fails, the RS MUST consider the Access Token non valid, and MUST respond to the Client with an error response code equivalent to the CoAP code 4.00 (Bad Request).

If the Access Token is valid and further checks on its content are successful, the RS proceeds as follows.

In case the POST request to /authz-info was not protected, the RS MUST generate a nonce N2, an OSCORE Recipient ID (ID2), and include them in the 2.01 (Created) response to the Client, as defined in the OSCORE profile of ACE (see Section 4.2 of [I-D.ietf-ace-oscore-profile]).

Instead, in case the POST request to /authz-info was protected, the RS MUST ignore any nonce and identifiers in the request, if any was sent. Then, the RS MUST check that the 'kid' field of the 'cnf' claim in the new Access Token matches the identifier of the OSCORE Input Material of a pairwise OSCORE Security Context such that:

- * The pairwise OSCORE Security Context was used to protect the request, if this was protected with OSCORE; or
- * The pairwise OSCORE Security Context is bound to the Group OSCORE Security Context used to protect the request, if this was protected with Group OSCORE.

If either verification is successful, the new Access Token supersedes the old one at the RS. Besides, the RS associates the new Access Token to the same pairwise OSCORE Security Context identified above, as also bound to a Group OSCORE Security Context. The RS MUST respond with a 2.01 (Created) response with no payload, protected with the same Security Context used to protect the request. In particular, no new pairwise OSCORE Security Context is established between the Client and the RS. If any verification fails, the RS MUST respond with a 4.01 (Unauthorized) error response.

Further recommendations, considerations and behaviors defined in Section 4.2 of [I-D.ietf-ace-oscore-profile] hold for this document.

A.3.3. OSCORE Setup - Client Side

Once having received the 2.01 (Created) response from the RS, following an unprotected POST request to the /authz-info endpoint, the Client MUST extract the nonce N2 from the 'nonce2' parameter, and the Client identifier from the 'ace_server_recipientid' parameter in the CBOR map of the response payload. Note that this identifier is used by C as Sender ID in the pairwise OSCORE Security Context to be established with the RS, and is different as well as unrelated to the Sender ID of C in the OSCORE group.

Then, the Client performs the following actions, in order to set up and fully derive the pairwise OSCORE Security Context for communicating with the RS.

- * The Client MUST set the ID Context of the pairwise OSCORE Security Context as the concatenation of: i) GID, i.e., the Group Identifier of the OSCORE group, as specified by the Client in the 'context_id' parameter of the Client-to-AS request; ii) the nonce N1; iii) the nonce N2; and iv) CID, i.e., the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' parameter of the Access Token response from the AS. The concatenation occurs in this order: ID Context = GID | N1 | N2 | CID, where | denotes byte string concatenation.
- * The Client MUST set the updated Master Salt of the pairwise OSCORE Security Context as the concatenation of SaltInput, MSalt, the nonce N1, the nonce N2 and GMSalt, where: i) SaltInput is the Sender ID that the Client has in the OSCORE group, which is known to the Client as a member of the OSCORE group; ii) MSalt is the (optional) Master Salt in the pairwise OSCORE Security Context (received from the AS in the Token); and iii) GMSalt is the (optional) Master Salt in the Group OSCORE Security Context, which is known to the Client as a member of the OSCORE group. The concatenation occurs in this order: Master Salt = SaltInput | MSalt | N1 | N2 | GMSalt, where | denotes byte string concatenation. Optional values, if not specified, are not included in the concatenation. The five parameters SaltInput, MSalt, N1, N2 and GMSalt are to be concatenated as encoded CBOR byte strings. An example of Master Salt construction using CBOR encoding is given in Figure 16.

SaltInput, MSalt, N1, N2 and GMSalt, in CBOR diagnostic notation:

```
SaltInput = h'00'
MSalt = h'f9af838368e353e78888e1426bd94e6f'
N1 = h'018a278f7faab55a'
N2 = h'25a8991cd700ac01'
GMSalt = h'99'
```

SaltInput, MSalt, N1, N2 and GMSalt, as CBOR encoded byte strings:

```
SaltInput = 0x4100
MSalt = 0x50f9af838368e353e78888e1426bd94e6f
N1 = 0x48018a278f7faab55a
N2 = 0x4825a8991cd700ac01
GMSalt = 0x4199
```

```
Master Salt = 0x41 00
               50 f9af838368e353e78888e1426bd94e6f
               48 018a278f7faab55a
               48 25a8991cd700ac01
               41 99
```

Figure 16: Example of Master Salt construction using CBOR encoding.

- * The Client MUST set the Master Secret of the pairwise OSCORE Security Context to the concatenation of MSec and GMSec, where: i) MSec is the value of the 'ms' parameter in the OSCORE_Input_Material of the 'cnf' parameter, received from the AS in Appendix A.2.2; while ii) GMSec is the Master Secret of the Group OSCORE Security Context, which is known to the Client as a member of the OSCORE group.
- * The Client MUST set the Recipient ID as ace_client_recipientid, sent as described in Appendix A.3.1.
- * The Client MUST set the Sender ID as ace_server_recipientid, received as described in Appendix A.3.1.
- * The Client MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version as indicated in the corresponding parameters received from the AS in Appendix A.2.2, if present in the OSCORE_Input_Material of the 'cnf' parameter. In case these parameters are omitted, the default values SHALL be used as described in Sections 3.2 and 5.4 of [RFC8613].

Finally, the client MUST derive the complete pairwise OSCORE Security Context following Section 3.2.1 of [RFC8613].

From then on, when communicating with the RS to access the resources as specified by the authorization information, the Client MUST use the newly established pairwise OSCORE Security Context or the Group OSCORE Security Context of the OSCORE Group where both the Client and the RS are members.

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

The Client can use this pairwise OSCORE Security Context to send requests to the RS protected with OSCORE. Besides, the Client can use the Group OSCORE Security Context for protecting unicast requests to the RS, or multicast requests to the OSCORE group including also the RS. Mutual authentication as group members is only achieved after the client has successfully verified the Group OSCORE protected response from the RS. Similarly, mutual authentication as OSCORE peers is only achieved after the client has successfully verified the OSCORE protected response from the RS, using the pairwise OSCORE Security Context.

Note that the ID Context of the pairwise OSCORE Security Context can be assigned by the AS, communicated and set in both the RS and Client after the exchange specified in this profile is executed. Subsequently, the Client and RS can update their ID Context by running a mechanism such as the one defined in Appendix B.2 of [RFC8613] if they both support it and are configured to do so. In that case, the ID Context in the pairwise OSCORE Security Context will not match the "contextId" parameter of the corresponding OSCORE_Input_Material. Running the procedure in Appendix B.2 of [RFC8613] results in the keying material in the pairwise OSCORE Security Contexts of the Client and RS being updated. The Client can achieve the same result by re-posting the Access Token to the unprotected /authz-info endpoint at the RS, as described in Section 4.1 of [I-D.ietf-ace-oscore-profile], although without updating the ID Context.

A.3.4. OSCORE Setup - Resource Server Side

After validation of the Access Token as defined in Appendix A.3.2 and after sending the 2.01 (Created) response to an unprotected POST request to the /authz-info endpoint, the RS performs the following actions, in order to set up and fully derive the pairwise OSCORE Security Context created to communicate with the Client.

- * The RS MUST set the ID Context of the pairwise OSCORE Security Context as the concatenation of: i) GID, i.e., the Group Identifier of the OSCORE group, as specified in the 'contextId' parameter of the OSCORE_Input_Material, in the 'cnf' claim of the Access Token received from the Client (see Appendix A.3.1); ii) the nonce N1; iii) the nonce N2; and iv) CID which is the value in the contextId parameter of the OSCORE_Input_Material provided in the 'cnf' claim of the Access Token. The concatenation occurs in this order: ID Context = GID | N1 | N2 | CID, where | denotes byte string concatenation.
- * The RS MUST set the new Master Salt of the pairwise OSCORE Security Context as the concatenation of SaltInput, MSalt, the nonce N1, the nonce N2 and GMSalt, where: i) SaltInput is the Sender ID that the Client has in the OSCORE group, as specified in the 'salt_input' claim included in the Access Token received from the Client (see Appendix A.3.1); ii) MSalt is the (optional) Master Salt in the pairwise OSCORE Security Context as specified in the 'salt' parameter in the OSCORE_Input_Material of the 'cnf' claim, included in the Access Token received from the Client; and iii) GMSalt is the (optional) Master Salt in the Group OSCORE Security Context, which is known to the RS as a member of the OSCORE group. The concatenation occurs in this order: Master Salt = SaltInput | MSalt | N1 | N2 | GMSalt, where | denotes byte string concatenation. Optional values, if not specified, are not included in the concatenation. The same considerations for building the Master Salt, considering the inputs as encoded CBOR byte strings as in Figure 16, hold also for the RS.
- * The RS MUST set the Master Secret of the pairwise OSCORE Security Context to the concatenation of MSec and GMSec, where: i) MSec is the value of the 'ms' parameter in the OSCORE_Input_Material of the 'cnf' claim, included in the Access Token received from the Client (see Appendix A.3.1); while ii) GMSec is the Master Secret of the Group OSCORE Security Context, which is known to the RS as a member of the OSCORE group.
- * The RS MUST set the Recipient ID as ace_server_recipientid, sent as described in Appendix A.3.2.
- * The RS MUST set the Sender ID as ace_client_recipientid, received as described in Appendix A.3.2.

- * The RS MUST set the AEAD Algorithm, ID Context, HKDF, and OSCORE Version from the corresponding parameters received from the Client in the Access Token (see Appendix A.3.1), if present in the OSCORE_Input_Material of the 'cnf' claim. In case these parameters are omitted, the default values SHALL be used as described in Sections 3.2 and 5.4 of [RFC8613].

Finally, the RS MUST derive the complete pairwise OSCORE Security Context following Section 3.2.1 of [RFC8613].

Once having completed the derivation above, the RS MUST associate the authorization information from the Access Token with the just established pairwise OSCORE Security Context. Furthermore, as defined in Section 4.2, the RS MUST associate the authorization information from the Access Token with the Group OSCORE Security Context.

Then, the RS uses this pairwise OSCORE Security Context to verify requests from and send responses to the Client protected with OSCORE, when this Security Context is used. If OSCORE verification fails, error responses are used, as specified in Section 8 of [RFC8613].

Besides, the RS uses the Group OSCORE Security Context to verify (multicast) requests from and send responses to the Client protected with Group OSCORE. When processing an incoming request protected with Group OSCORE, the RS MUST consider as valid authentication credential of the Client only the authentication credential associated with the stored Access Token. As defined in Appendix A.3.6, a change of authentication credential in the group requires the Client to upload to the RS a new Access Token, where the 'client_cred' claim specifies a COSE Key equivalent to the new authentication credential that the Client has in the group.

If Group OSCORE verification fails, error responses are used, as specified in Sections 8 and 9 of [I-D.ietf-core-oscore-groupcomm]. Additionally, for every incoming request, if OSCORE or Group OSCORE verification succeeds, the verification of access rights is performed as described in Appendix A.3.5.

After the deletion of the Access Token related to a pairwise OSCORE Security Context and to a Group OSCORE Security Context, due to, for example, expiration, the RS MUST NOT use the pairwise OSCORE Security Context. The RS MUST respond with an unprotected 4.01 (Unauthorized) error message to received requests that correspond to a pairwise OSCORE Security Context with a deleted Access Token. Also, if the Client uses the Group OSCORE Security Context to send a request for any resource intended for OSCORE group members and that requires an active Access Token, the RS MUST respond with a 4.01 (Unauthorized) error message protected with the Group OSCORE Security Context.

The same considerations, related to the value of the ID Context changing, as in Appendix A.3.3 hold also for the RS.

A.3.5. Access Rights Verification

The RS MUST follow the procedures defined in Section 4.4.

Additionally, if the RS receives an OSCORE-protected request from a Client, the RS processes it according to [RFC8613].

If the OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the authorization information from the Access Token associated with the pairwise OSCORE Security Context and to the Group OSCORE Security Context. Then, the RS MUST verify that the action requested on the resource is authorized.

The response code MUST be 4.01 (Unauthorized) if the RS has no valid Access Token for the Client.

A.3.6. Change of Client's Authentication Credential in the Group

During its membership in the OSCORE group, the client might change the authentication credential it uses in the group. When this happens, the Client uploads the new authentication credential to the Group Manager, as defined in Section 11 of [I-D.ietf-ace-key-groupcomm-oscure].

After that, the Client may still have an Access Token previously uploaded to the RS, which is not expired yet and still valid to the best of the Client's knowledge. Then, in order to continue communicating with the RS, the Client MUST perform the following actions.

1. The Client requests a new Access Token to the AS, as defined in Appendix A.2.1 for the update of access rights, i.e., with the 'req_cnf' parameter including only a 'kid' field. In particular, when sending the POST request to the AS, the Client indicates:

- * The current Group Identifier of the OSCORE group, as value of the 'context_id' parameter.
 - * The current Sender ID it has in the OSCORE group, as value of the 'salt_input' parameter.
 - * The public key of the new authentication credential it uses in the OSCORE group, as value of the 'client_cred' parameter. In particular, the specified public key is the COSE Key equivalent to the new authentication credential that the Client uses in the OSCORE group.
 - * The proof-of-possession (PoP) evidence corresponding to the public key of the new authentication credential, as value of the 'client_cred_verify' or 'client_cred_verify_mac' parameter.
 - * The same current or instead new set of access rights, as value of the 'scope' parameter.
2. After receiving the response from the AS (see Appendix A.2.2), the Client performs the same exchanges with the RS as defined in Appendix A.3, with the following difference: the POST request to /authz-info for uploading the new Access Token MUST be protected with the pairwise OSCORE Security Context shared with the RS.

When receiving the new Access Token, the RS performs the same steps defined in Appendix A.3.2. In particular, no new pairwise OSCORE Security Context is established between the Client and the RS.

A.4. Secure Communication with the AS

The same considerations for secure communication with the AS as defined in Section 5 hold.

A.5. Discarding the Security Context

The Client and the RS MUST follow what is defined in Section 6 of [I-D.ietf-ace-oscore-profile] about discarding the pairwise OSCORE Security Context.

Additionally, they MUST follow what is defined in the main mode of the profile (see Section 6), with respect to the Group OSCORE Security Context.

The Client or RS can acquire a new Group OSCORE Security Context, by re-joining the OSCORE group, e.g., by using the approach defined in [I-D.ietf-ace-key-groupcomm-oscore]. In such a case, the Client

SHOULD request a new Access Token and post it to the RS, in order to establish a new pairwise OSCORE Security Context and bind it to the Group OSCORE Security Context obtained upon re-joining the group.

A.6. CBOR Mappings

The new parameters defined in this document MUST be mapped to CBOR types as specified in Figure 6, with the following addition, using the given integer abbreviation for the map key.

Parameter name	CBOR Key	Value Type
client_cred	TBD	map

Figure 17: CBOR mappings for new parameters.

The new claims defined in this document MUST be mapped to CBOR types as specified in Figure 7, with the following addition, using the given integer abbreviation for the map key.

Claim name	CBOR Key	Value Type
client_cred	TBD	map

Figure 18: CBOR mappings for new claims.

A.7. Security Considerations

The dual mode of this profile inherits the security considerations from the main mode (see Section 8), as well as from the security considerations of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile]. Also, the security considerations about OSCORE [RFC8613] hold for the dual mode of this profile, as to the specific use of OSCORE.

Unlike the main mode and consistently with Section 6.1 of [I-D.ietf-ace-oauth-authz], the dual mode of this profile cannot be used to issue an Access Token for an audience that comprises multiple RSs. This is because the proof-of-possession key bound to an Access Token is the OSCORE Master Secret included in the OSCORE_Input_Material object of the 'cnf' claim, and it has to be shared only between the Client and one RS.

A.8. Privacy Considerations

The same privacy considerations as defined in the main mode of this profile apply (see Section 9).

In addition, as this profile mode also uses OSCORE, the privacy considerations from [RFC8613] apply as well, as to the specific use of OSCORE.

Furthermore, this profile mode inherits the privacy considerations from the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

Appendix B. Profile Requirements

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

- * (Optional) discovery process of how the Client finds the right AS for an RS it wants to send a request to: Not specified.
- * Communication protocol the Client and the RS must use: CoAP.
- * Security protocol(s) the Client and RS must use: Group OSCORE, i.e., exchange of secure messages by using a pre-established Group OSCORE Security Context. The optional dual mode defined in Appendix A additionally uses OSCORE, i.e., establishment of a pairwise OSCORE Security Context and exchange of secure messages.
- * How the Client and the RS mutually authenticate: Explicitly, by possession of a common Group OSCORE Security Context, and by either: usage of digital signatures embedded in messages, if protected with the group mode of Group OSCORE; or protection of messages with the pairwise mode of Group OSCORE, by using pairwise symmetric keys, derived from the asymmetric keys of the two peers exchanging the message. Note that the mutual authentication is not completed before the Client has verified an OSCORE or a Group OSCORE response using the corresponding security context.
- * Content-format of the protocol messages: "application/ace+cbor".
- * Proof-of-Possession protocol(s) and how to select one; which key types (e.g., symmetric/asymmetric) supported: Group OSCORE algorithms; distributed and verified asymmetric keys. In the optional dual mode defined in Appendix A: OSCORE algorithms; pre-established symmetric keys.
- * profile identifier: coap_group_oscore

- * (Optional) how the RS talks to the AS for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE).
- * How the client talks to the AS for requesting a token: HTTP/CoAP (+ TLS/DTLS/OSCORE).
- * How/if the authz-info endpoint is protected: Not protected.
- * (Optional) other methods of token transport than the authz-info endpoint: Not specified.

Acknowledgments

The authors sincerely thank Benjamin Kaduk, John Mattsson, Dave Robin, Jim Schaad and Goeran Selander for their comments and feedback.

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

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: rikard.hoglund@ri.se

Ludwig Seitz
Combitech
Djäcknegatan 31
SE-21135 Malmö Malmö
Sweden
Email: ludwig.seitz@combitech.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: francesca.palombini@ericsson.com