ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 13 January 2022

F. Palombini
Ericsson AB
M. Tiloca
RISE AB

12 July 2021

Key Provisioning for Group Communication using ACE draft-ietf-ace-key-groupcomm-13

Abstract

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

Discussion Venues

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 13 January 2022.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Terminology 4
2. Overview 5
3. Authorization to Join a Group 8
3.1. Authorization Request 8
3.2. Authorization Response 10
3.3. Token Post 12
4. Keying Material Provisioning and Group Membership Management 15
4.1. Interface at the KDC 17
4.2. Retrieval of Group Names and URIs 39
4.3. Joining Exchange 40
4.4. Retrieval of Updated Keying Material 42
4.5. Requesting a Change of Keying Material 44
4.6. Retrieval of Public Keys and Roles for Group Members . . 45
4.7. Update of Public Key 47
4.8. Retrieval of Group Policies 48
4.9. Retrieval of Keying Material Version 49
4.10. Group Leaving Request 50
5. Removal of a Node from the Group 50
6. Extended Scope Format 52
7. ACE Groupcomm Parameters 54
8. ACE Groupcomm Error Identifiers 55
9. Security Considerations 56
9.1. Update of Keying Material 57
9.2. Block-Wise Considerations 58
10. IANA Considerations 58
10.1. Media Type Registrations 58
10.2. CoAP Content-Formats Registry 59
10.3. OAuth Parameters Registry 60
10.4. OAuth Parameters CBOR Mappings Registry 60
10.5. ACE Groupcomm Parameters Registry 61
10.6. ACE Groupcomm Key Registry 61
10.7. ACE Groupcomm Profile Registry 62
10.8. ACE Groupcomm Policy Registry 63
10.9. Sequence Number Synchronization Method Registry 63
10.10. Interface Description (if=) Link Target Attribute Values Registry 64
10.11. CBOR Tags Registry 64
10.12. ACE Scope Semantics 65

10.13. ACE Groupcomm Errors 65
10.14. Expert Review Instructions 66
11. References 66
11.1. Normative References 66
11.2. Informative References 69
Appendix A. Requirements on Application Profiles 70
Appendix B. Extensibility for Future COSE Algorithms 73
B.1. Format of 'sign_info_entry' 74
Appendix C. Document Updates 74
C.1. Version -04 to -05 74
C.2. Version -03 to -04 75
C.3. Version -02 to -03 75
C.4. Version -01 to -02 76
C.5. Version -00 to -01 76
Acknowledgments 77
Authors' Addresses 77

1. Introduction

This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to define the message exchanges used to request, distribute and renew the keying material in a group communication scenario, e.g., based on multicast [I-D.ietf-core-groupcomm-bis] or on publishing-subscribing [I-D.ietf-core-coap-pubsub]. The ACE framework is based on CBOR [RFC8949], so CBOR is the format used in this specification. However, using JSON [RFC8259] instead of CBOR is possible, using the conversion method specified in Sections 6.1 and 6.2 of [RFC8949].

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

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

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz][I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], such as Authorization Server (AS) and Resource Server (RS).

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

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

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

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

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

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

This document additionally uses the following terminology:

* Transport profile, to indicate a profile of ACE as per Section 5.8.4.3 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. Tranport profiles of ACE include, for instance, [I-D.ietf-ace-oscore-profile], [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-mqtt-tls-profile].

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

2. Overview

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

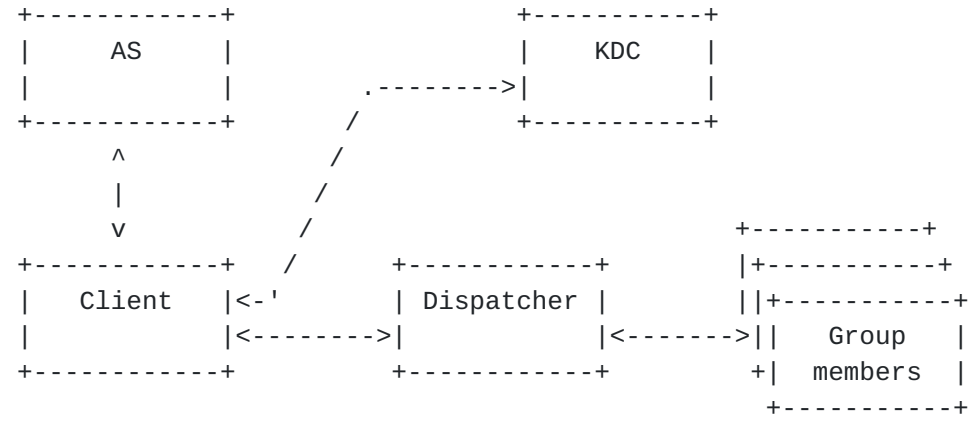

Figure 1: Key Distribution Participants

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

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

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

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

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

This document specifies a mechanism for:

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

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

* Allowing a group member to retrieve public keys of other group members (Section 4.6) and to provide an updated public key (Section 4.7).

* 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 4.10 and Section 5).

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

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

2. The joining node transfers authentication and authorization information to the KDC, by posting the obtained Access Token to the /authz-info endpoint at the KDC. This exchange, and all further communications between the Client and the KDC, MUST occur over the secure channel established as a result of the transport profile of ACE used between Client and KDC. After that, a

joining node MUST have a secure communication association established with the KDC, before starting to join a group under that KDC. Possible ways to provide a secure communication association are described in the DTLS transport profile [I-D.ietf-ace-dtls-authorize] and OSCORE transport profile [I-D.ietf-ace-oscore-profile] of ACE.

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

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

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

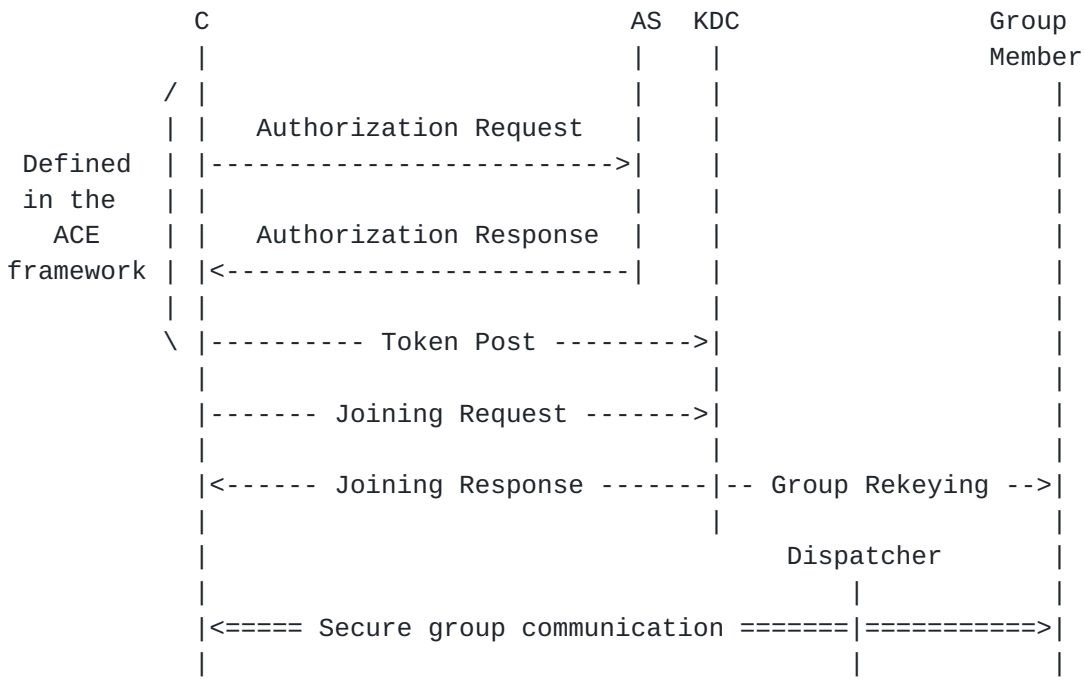

Figure 2: Message Flow Upon New Node's Joining

3. Authorization to Join a Group

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

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

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

Figure 3 gives an overview of the exchange described above.

Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

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

* 'scope', containing the identifier of the specific groups, or topics in the case of pub-sub, that the Client wishes to access, and optionally the roles that the Client wishes to take.

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

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

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

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

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

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

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

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

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

```
gname = tstr

permissions = uint . bits roles

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

scope_entry = AIF_Generic<gname, permissions>

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

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

```
gname = tstr

role = tstr

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

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

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

3.2. Authorization Response

The Authorization Response sent from the AS to the Client is defined in Section 5.8.2 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 6 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 6.

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

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

3.3. Token Post

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

This request differs from the one defined in [I-D.ietf-ace-oauth-authz], because it allows to transport additional encoding information about the public keys in the group, used for source authentication, as well as any other group parameters.

The joining node MAY ask for this information from the KDC in the same message it uses to POST the token to the RS. In such a case, the message MUST have Content-Format set to application/ace+cbor defined in Section 8.16 of [I-D.ietf-ace-oauth-authz]. 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 the corresponding values:

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

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

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

The KDC replies to the Client with a 2.01 (Created) response, using Content-Format "application/ace+cbor".

The payload of the 2.01 response is a CBOR map. If the access token contains a role that requires the Client to send its own public key to the KDC when joining the group, the CBOR map MUST include the parameter 'kdcchallenge' defined in Section 3.3.2, specifying a dedicated challenge N_S generated by the KDC. The Client uses this challenge to prove possession of its own private key (see the 'client_cred_verify' parameter in Section 4). Note that the payload format of the response deviates from the one defined in the ACE framework (see Section 5.10.1 of [I-D.ietf-ace-oauth-authz]), which has no payload.

The KDC MUST store the 'kdcchallenge' value associated to the Client at least until it receives a join request from it (see Section 4.3), to be able to verify that the Client possesses its own private key. The same challenge MAY be reused several times by the Client, to generate a new proof of possession, e.g., in case of update of the

public key, or to join a different group with a different signing key, so it is RECOMMENDED that the KDC keeps storing the 'kdcchallenge' after the first join is processed as well. If the KDC has already discarded the 'kdcchallenge', that will trigger an error response with a newly generated 'kdcchallenge' that the Client can use to restart the join process, as specified in Section 4.3.

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

Note that the CBOR map specified as payload of the 2.01 (Created) response may include further parameters, e.g. according to the signalled transport profile of ACE. Application profiles MAY define the additional parameters to use within this exchange (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 (OPT1).

3.3.1. 'sign_info' Parameter

The 'sign_info' parameter is an OPTIONAL parameter of the Token Post response message defined in Section 5.10.1. of [I-D.ietf-ace-oauth-authz]. This parameter contains information and parameters about the signature algorithm and the public keys to be used between the Client and the RS. Its exact content is application specific.

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

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

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

```
sign_info_req = nil
```

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

* The first element 'id' is a group name or an array of group names, 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 Null, 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 Null 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 formatted as in the response is given below.

```
sign_info_res = [ + sign_info_entry ]

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

gname = tstr
```

This format is consistent with every signature algorithm currently considered 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 the Token Post response message defined in Section 5.10.1 of [I-D.ietf-ace-oauth-authz]. This parameter contains a challenge generated by the KDC and provided to the Client. The Client may use this challenge to prove possession of its own private key in the Joining Request (see the 'client_cred_verify' parameter in Section 4).

4. Keying Material Provisioning and Group Membership Management

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

During the first exchange with the KDC ("Joining") after posting the Token, the Client sends a request to the KDC, specifying the group it wishes to join (see Section 4.3). Then, the KDC verifies the access token and that the Client is authorized to join that group. If so, it provides the Client with the keying material to securely communicate with the other members of the group.

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

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

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

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

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

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

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

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

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

If they include a payload and specify a Content-Format, requests sent to the KDC and success responses from the KDC MUST have Content-Format set to application/ace-groupcomm+cbor, defined in Section 10.2.

Some error responses from the KDC can have Content-Format set to application/ace-groupcomm+cbor. In such a case, the paylod 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 10.13 of this specification. This field MUST be present.

* 'error_description', with value a CBOR text string specifying a human-readable description of the error occurred at the KDC. This field MAY be present.

CBOR labels for the 'error' and 'error_description' fields are defined in Section 7.

Section 8 of this specification defines an initial set of error identifiers, as possible values for the 'error' field. Application profiles of this specification MAY define additional value (OPT11).

4.1. Interface at the KDC

The KDC is configured with the following resources. Note that the root url-path "ace-group" given here are default names: implementations are not required to use these names, and can define their own instead. Each application profile of this specification MUST register a Resource Type for the root url-path (REQ7), and that Resource Type can be used to discover the correct url to access at the KDC. This Resource Type can also be used at the GROUPNAME sub-resource, to indicate different application profiles for different groups. The Interface Description (if=) Link Target Attribute value ace.group is registered (Section 10.10) and can be used to describe this interface.

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

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

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

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

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

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

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

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

It is REQUIRED of the application profiles of this specification to define what operations (e.g., CoAP methods) are allowed on each resource, for each role defined in Section 3.1 according to REQ2 (REQ8).

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

4.1.1. ace-group

This resource implements a FETCH handler.

4.1.1.1. FETCH Handler

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

The handler expects a request with payload formatted as a CBOR map, 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 (REQ9). The Client indicates that it wishes to receive the group names and GROUPNAMEs of all groups having these identifiers.

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

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

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

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

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

If the KDC does not find any group associated 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.

4.1.2. ace-group/GROUPNAME

This resource implements GET and POST handlers.

4.1.2.1. POST Handler

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

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

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

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

  The parameter's value is either the CBOR simple value Null, or a non-empty CBOR array containing the following three elements.

  - The first element, namely 'inclusion_flag', encodes the CBOR simple value True.

  - 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. The Client indicates that it wishes to receive the public keys of all group members having any of the single roles, or at least all of the roles indicated in any 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.

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

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

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

Also note that the array of node identifiers 'id_filter' is empty for this handler, because the joining node is not expected or capable to express a filter based on node identifiers at this point in time. Consistently, the 'inclusion_flag' element is set to the CBOR simple value True. However, the 'id_filter' array is not necessarily empty for the value of 'get_pub_keys' received by the handler of FETCH to ace-group/GROUPNAME/pub-key (see Section 4.1.3.1).

Finally, the 'get_pub_keys' parameter MUST NOT have the arrays 'role_filter' and 'id_filter' as both empty, i.e., in CBOR diagnostic notation: [bool, [ ], [ ]]. Thus, if this parameter is received as formatted in that way, it has to be considered malformed.

```
id = bstr

role = tstr

comb_role = [ 2*role ]

inclusion = bool

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

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

* 'client_cred', encoded as a CBOR byte string, which wraps 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).

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

* 'client_cred_verify', encoded as a CBOR byte string. This parameter MUST be present if the 'client_cred' parameter is present and no public key associated to the client's token can be retrieved for that group.

This parameter contains a 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 POST request (see Section 3.3), encoded as a CBOR byte string.

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

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

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 (REQ20).

If the token was not posted (e.g., if it is used directly to validate TLS instead), it is REQUIRED of the specific profile to define how the challenge N_S is generated (REQ21).

* '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 (OPT3).

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

mechanisms using this resource, it can just ignore this parameter. Other additional functionalities of this resource MAY be defined in application profiles of this specifications (OPT9). In particular, this resource is intended for communications concerning exclusively the group or topic specified in the 'scope' parameter.

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


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

  PoP input =
    0x4f 826667726F7570316673656E646572
      48 018a278f7faab55a 48 25a8991cd700ac01
```

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

The handler extracts the granted scope from the access token, and checks the requested one against the token one. If the requested one is not a subset of the token one, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If the request does not include a 'scope' field, the KDC is expected to understand which group and role(s) the Client is requesting (e.g., there is only one the Client has been granted). If the KDC can not recognize which scope the Client is requesting, it MUST respond with a 4.00 (Bad Request) error message.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 (i.e., required fields non received or received with incorrect format), the handler MUST respond with a 4.00 (Bad Request) error message. The response MAY 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 Null if the Client sent its own public key and the KDC is not set to store public keys of the group members.

If the request contained unknown or non-expected fields present, the handler MUST silently drop them and continue processing. Application profiles MAY define optional or mandatory payload formats for specific error cases (OPT5).

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 posted but the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST respond with a 4.00 Bad Request error response, 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, whose CBOR label is defined in Section 7.

* 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 respond with a 4.00 (Bad Request) error message. 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 (REQ20).

If the PoP evidence does not pass verification, the handler MUST respond with a 4.01 (Unauthorized) error message. 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 one public key is already associated to the received access token (see Section 4.3 for an example) and to the group identified by GROUPNAME.

If an eligible public key for the Client is neither present in the 'client_cred' field nor already stored, it is RECOMMENDED that the handler stops the processing and responds with a 4.00 (Bad Request) error message. Applications profiles MAY define alternatives (OPT6).

If all the verifications above succeed, 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 identifier NODENAME to the Client, and creates a sub-resource to /ace-group/GROUPNAME/ at the KDC (e.g., "/ace-group/GROUPNAME/nodes/NODENAME").

* The handler associates the node identifier NODENAME to the access token and the secure session for the Client.

* If the KDC manages the group members' public keys:

  - The handler associates the retrieved Client's public key to the node identifier NODENAME and to the 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.

* The handler returns a 2.01 (Created) response, 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 response message also contains the URI path to the sub-resource created for that node in a Location-Path CoAP option. The response 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" Registry. Implementations MUST verify that the key type matches the application profile being used, if present, as registered in the "ACE Groupcomm Key" registry.

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

* 'num', containing the version number of the keying material for the group communication, formatted as an integer. This is a strictly monotonic increasing field. The application profile MUST define the initial version number (REQ23).

The exact format of the 'key' value MUST be defined in applications of this specification (REQ10), as well as values of 'gkty' accepted by the application (REQ11). Additionally, documents specifying the key format MUST register it in the "ACE Groupcomm Key" registry defined in Section 10.6, including its name, type and application profile to be used with.

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

Figure 8: Key Type Values

The response SHOULD contain the following parameter:

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

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

* 'ace-groupcomm-profile', with value a CBOR integer that MUST be used to uniquely identify the application profile for group communication. Applications of this specification MUST register an application profile identifier and the related value for this parameter in the "ACE Groupcomm Profile" Registry (REQ15).

* '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, which wraps 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 wrapping 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 Policy" Registry. This specification defines the three elements "Sequence Number Synchronization Method", "Key Update Check Interval" and "Expiration Delta", which are summarized in Figure 9. Application profiles that build on this document MUST specify the exact content format and default value of included map entries (REQ17).

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

Figure 9: ACE Groupcomm Policies

* 'mgt_key_material', encoded as a CBOR byte string and containing the administrative keying material to participate in the group rekeying performed by the KDC. The application profile MUST define if this field is used, and if used then MUST specify the exact format and content which depend on the specific rekeying scheme used in the group. If the usage of 'mgt_key_material' is indicated and its format defined for a specific key management scheme, that format must explicitly indicate the key management scheme itself. If a new rekeying scheme is defined to be used for an existing 'mgt_key_material' in an existing profile, then that profile will have to be updated accordingly, especially with respect to the usage of 'mgt_key_material' related format and content (REQ22).

Specific application profiles that build on this document MUST specify the communication protocol that members of the group use to communicate with each other (REQ13) and how exactly the keying material is used to protect the group communication (REQ14).

CBOR labels for these fields are defined in Section 7.

4.1.2.2. GET Handler

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

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, the handler returns a 2.05 (Content) message containing the symmetric group keying material. The payload of the response is formatted as a CBOR map which MUST contain the parameters 'gkty', 'key' and 'num' specified in Section 4.1.2.1.

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

4.1.3. ace-group/GROUPNAME/pub-key

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

This resource implements GET and FETCH handlers.

4.1.3.1. FETCH Handler

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

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

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

  - The 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'. 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'.

  - The array 'role_filter' may 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' parameter encodes the CBOR simple value True or False, respectively. The array 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 both the 'role_filter' array and the 'id_filter' array are not 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'.

Finally, as mentioned in Section 4.1.2.1, both arrays 'role_filter' and 'id_filter' MUST NOT be both empty.

The specific format of public keys as well as identifiers, roles and combination of roles of group members MUST be specified by the application profile (OPT1, REQ2, REQ12).

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

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

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

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

Then, the handler returns a 2.05 (Content) message response with payload formatted as a CBOR map, containing only the following parameters from Section 4.1.2.1.

* '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, REQ12).

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 (REQ16).

* The KDC silently ignores those node identifiers.

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

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

4.1.3.2. GET Handler

The handler expects a GET request.

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

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

4.1.4. ace-group/GROUPNAME/policies

This resource implements a GET handler.

4.1.4.1. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, the handler returns a 2.05 (Content) message containing the list of policies for the group identified by GROUPNAME. The payload of the response is formatted as a CBOR map including only the parameter 'group_policies' defined in Section 4.1.2.1 and specifying the current policies in the group. If the KDC does not store any policy, the payload is formatted as a zero-length CBOR byte string.

The specific format and meaning of group policies MUST be specified in the application profile (REQ17).

4.1.5. ace-group/GROUPNAME/num

This resource implements a GET handler.

4.1.5.1. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to this client. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, the handler returns a 2.05 (Content) message containing an integer that represents the version number of the symmetric group keying material. This number is incremented on the KDC every time the KDC updates the symmetric group keying material, before the new keying material is distributed. This number is stored in persistent storage.

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

4.1.6. ace-group/GROUPNAME/nodes/NODENAME

This resource implements GET, PUT and DELETE handlers.

4.1.6.1. PUT Handler

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

The handler expects a request with empty payload. In case the request has a non-empty payload, the KDC MUST respond with a 4.00 (Bad Request) error message.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to the client identified by NODENAME. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

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

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

Also, 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 respond with a 4.00 (Bad Request) error message. 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 respond with a 5.03 (Service Unavailable) error message. 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 returns a 2.05 (Content) message containing newly-generated keying material for the Client, and/or, if the application profiles requires it (OPT8), starts the complete group rekeying. The payload of the response is formatted as a CBOR map. The specific format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ18) and registered in Section 10.5.

4.1.6.2. GET Handler

The handler expects a GET request.

The KDC verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to the client identified by NODENAME. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

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

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, the handler returns a 2.05 (Content) message containing both the group keying material and the individual keying material for the Client, or information enabling the Client to derive it. The payload of the response is formatted as a CBOR map. The format for the group keying material is the same as defined in the response of Section 4.1.2.2. The specific format of individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ18) and registered in Section 10.5.

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).

4.1.6.3. DELETE Handler

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

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

The handler verifies that the group name of the /ace-group/GROUPNAME path is a subset of the 'scope' stored in the access token associated to the client identified by NODENAME. If the verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

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

Additionally, the handler verifies that the node is a current member of the group. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, the handler removes the client from the group identified by GROUPNAME. That includes removing the public key of the client if the KDC keep tracks of that, and possibly removing the evicted node from the list of observers of the resource at ace-group/GROUPNAME (if observable). Then, the handler deletes the sub-resource nodes/NODENAME and returns a 2.02 (Deleted) message with empty payload.

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

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

4.1.7.1. POST Handler

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

The handler expects a POST request with payload as specified in Section 4.1.2.1, with the difference that it includes only the parameters 'client_cred', 'cnonce' and 'client_cred_verify'. In particular, the PoP evidence included in 'client_cred_verify' is computed in the same way considered in Section 4.1.2.1 and defined by the specific application profile (REQ20), 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).

The handler verifies that the group name GROUPNAME is a subset of the 'scope' stored in the access token associated to the client identified by NODENAME. The KDC also verifies that the roles the client is granted in the group allow it to perform this operation on this resource (REQ8). If either verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

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

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

Also, 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 respond with a 4.00 (Bad Request) error message. 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 request is not formatted correctly (i.e., required fields non received or received with incorrect format), the handler MUST respond with a 4.00 (Bad Request) error message. If the request contains unknown or non-expected fields present, the handler MUST silently ignore them and continue processing. Application profiles MAY define optional or mandatory payload formats for specific error cases (OPT5).

If the KDC cannot retrieve the 'kdcchallenge' associated to this Client (see Section 3.3), the KDC MUST respond with a 4.00 Bad Request error response, 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, whose CBOR label is defined in Section 7. 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 respond with a 4.00 (Bad Request) error message. 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 4.1.2.1 and defined by the specific application profile (REQ20). If the PoP evidence does not pass verification, the handler MUST respond with a 4.01 (Unauthorized) error message. 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 verification succeeds, 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.

4.2. Retrieval of Group Names and URIs

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

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

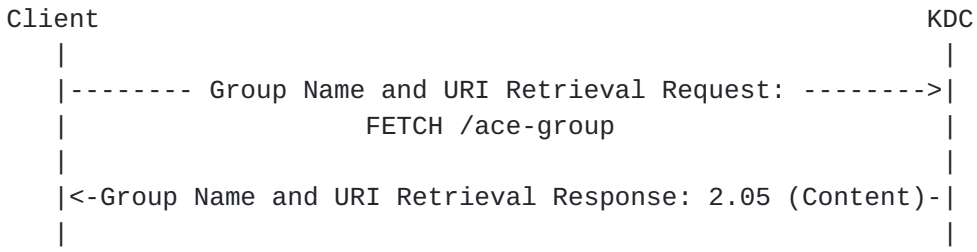

Figure 10: Message Flow of Group Name and URI Retrieval Request-Response

Request:

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

Response:

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

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

4.3. Joining Exchange

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

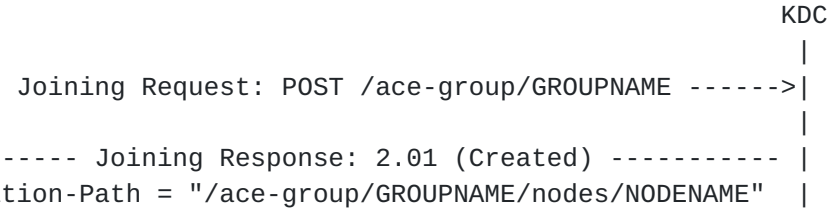

Figure 12: Message Flow of First Exchange for Group Joining

```
Request:

Header: POST (Code=0.02)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation,
         with PUB_KEY and 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 13: Example of First Exchange for Group Joining

If not previously established, the Client and the KDC MUST first establish a pairwise secure communication channel (REQ19). This can be achieved, for instance, by using a transport profile of ACE. The Joining exchange MUST occur over that secure channel. The Client and the KDC MAY use that same secure channel to protect further pairwise communications that must be secured.

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

To join the group, the Client sends a CoAP POST request to the /ace-group/GROUPNAME endpoint at the KDC, where GROUPNAME is the group name of the group to join, formatted as specified in Section 4.1.2.1. This group name is the same as in the scope entry corresponding to that group, specified in the 'scope' parameter of the Authorization Request/Response, or it can be retrieved from it. Note that, in case of successful joining, the Client will receive the URI to retrieve group keying material and to leave the group in the Location-Path option of the response.

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

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

4.4. Retrieval of Updated Keying Material

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

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

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

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

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

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

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

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

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

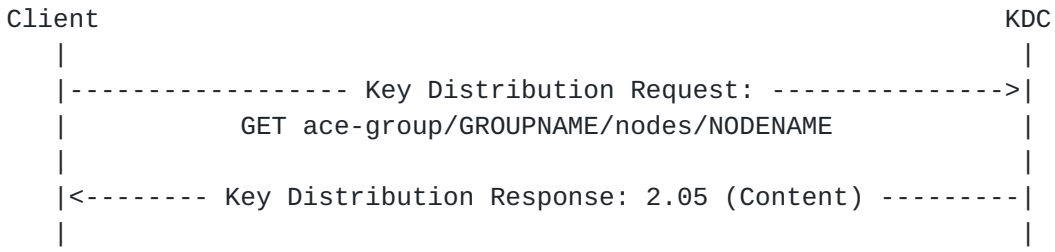

Figure 14: Message Flow of Key Distribution Request-Response

Request:

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

Response:

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

Figure 15: Example of Key Distribution Request-Response

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

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

In case the KDC deletes the group identified by GROUPNAME, 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").

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

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

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

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

4.5. Requesting a Change of Keying Material

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

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

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

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

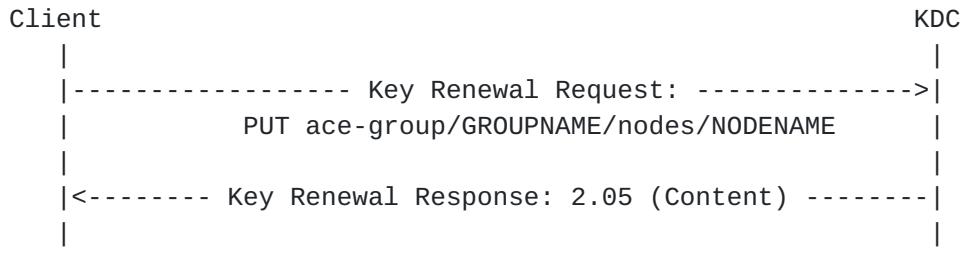

Figure 16: Message Flow of Key Renewal Request-Response

```
Request:

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

Response:

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

Figure 17: Example of Key Renewal Request-Response

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

4.6. Retrieval of Public Keys and Roles for Group Members

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

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

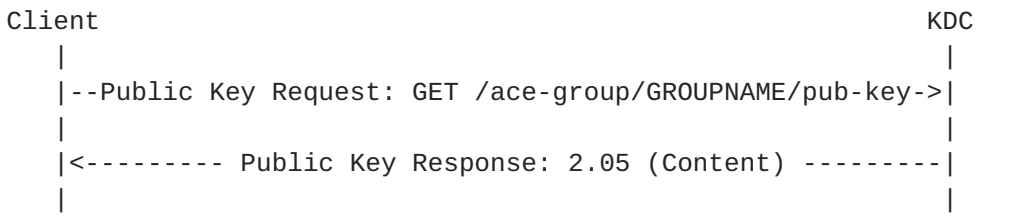

Figure 18: Message Flow of Public Key Exchange to Request All Members Public Keys

Request:

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

Response:

```
Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
  { "num": 5,
    "pub_keys": [ PUB_KEY1, PUB_KEY2, PUB_KEY3 ],
    "peer_roles": ["sender", ["sender", "receiver"], "receiver"],
    "peer_identifiers": [ ID1, ID2, ID3 ] }
```

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

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

```
            Request:

            Header: FETCH (Code=0.05)
            Uri-Host: "kdc.example.com"
            Uri-Path: "ace-group"
            Uri-Path: "g1"
            Uri-Path: "pub-key"
            Content-Format: "application/ace-groupcomm+cbor"
            Payload:
              { "get_pub_keys": [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 21: Example of Public Key Exchange to Request Specific Members Public Keys

4.7. Update of Public Key

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

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

The request is formatted as specified in Section 4.1.7.1.

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

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

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

```
Request:

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

Response:

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

Figure 23: Example of Public Key Update Request-Response

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

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

4.8. Retrieval of Group Policies

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

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

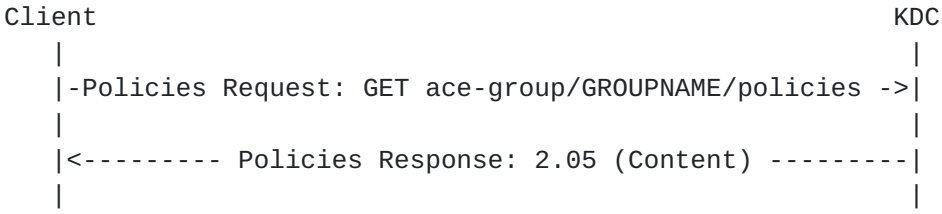

Figure 24: Message Flow of Policies Request-Response

```
Request:

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

Response:

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

Figure 25: Example of Policies Request-Response

4.9. Retrieval of Keying Material Version

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

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

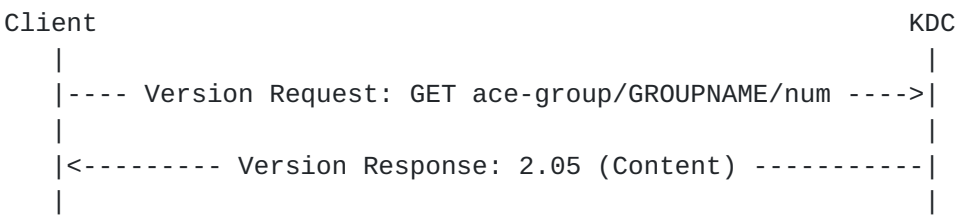

Figure 26: Message Flow of Version Request-Response

```
Request:

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

Response:

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

Figure 27: Example of Version Request-Response

4.10. Group Leaving Request

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

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

5. Removal of a Node from the Group

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

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

increment the version number of the current keying material, before distributing the newly generated keying material to the group. After that, the KDC SHOULD store the distributed keying material in persistent storage.

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

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

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

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

3. The node's authorization to be a group member is not valid anymore, either because the access token has expired, or it has been revoked. If the AS provides Token introspection (see Section 5.9 of [I-D.ietf-ace-oauth-authz]), the KDC can optionally use it and check whether the node is still authorized for that group in that role.

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

Furthermore, in case of forced eviction, the KDC removes the public key of the evicted node if the KDC keep tracks of that, and possibly removes the evicted node from the list of observers of the resource at ace-group/GROUPNAME (if observable).

Then, the KDC deletes the sub-resource ace-group/GROUPNAME/nodes/NODENAME associated to the evicted node. After that, the KDC MAY explicitly inform the evicted node, by means of the following methods.

* If the evicted node implements the 'control_uri' resource specified in Section 4.1.2.1, the KDC sends a DELETE request, targeting the URI specified in the 'control_uri' parameter of the Joining Request (see Section 4.1.2.1).

* If the evicted node is observing its associated sub-resource at ace-group/GROUPNAME/nodes/NODENAME (see Section 4.1.6.2), the KDC sends an unsolicited 4.04 (Not Found) 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").

  Consistently, the KDC also removes the node's entry from the list of observers of the sub-resource.

6. 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 10.12 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 (REQ24).

* 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 10.11 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 28 and Figure 29 build on the examples in Section 3.2, and show the corresponding extended scopes.

```
gname = tstr

permissions = uint . bits roles

roles = &(
   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 28: 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 29: CDLL definition of scope, using as example group name encoded as tstr and role as tstr

7. ACE Groupcomm Parameters

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

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

Name	CBOR Key	CBOR Type	Reference
error	TBD	int	Section 4
error_description	TBD	text string	Section 4
scope	TBD	byte string	Section 4.1.2.1
get_pub_keys	TBD	array / simple value null	Section 4.1.2.1, Section 4.1.3.1
client_cred	TBD	byte string	Section 4.1.2.1
cnonce	TBD	byte string	Section 4.1.2.1
client_cred_verify	TBD	byte string	Section 4.1.2.1

pub_keys_repos	TBD	text string	Section 4.1.2.1	
control_uri	TBD	text string	Section 4.1.2.1	
gkty	TBD	integer / text string	Section 4.1.2.1	
key	TBD	see "ACE Groupcomm Key" Registry	Section 4.1.2.1	
num	TBD	int	Section 4.1.2.1	
ace-groupcomm-profile	TBD	int	Section 4.1.2.1	
exp	TBD	int	Section 4.1.2.1	
pub_keys	TBD	array	Section 4.1.2.1	
peer_roles	TBD	array	Section 4.1.2.1	
peer_identifiers	TBD	array	Section 4.1.2.1	
group_policies	TBD	map	Section 4.1.2.1	
mgt_key_material	TBD	byte string	Section 4.1.2.1	
sign_info	TBD	array	Section 4.1.2.1	
gid	TBD	array	Section 4.1.1.1	
gname	TBD	array of text strings	Section 4.1.1.1	
guri	TBD	array of text strings	Section 4.1.1.1	
kdcchallenge	TBD	byte string	Section 4.1.7.1	

Table 1

8. 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

Table 2

9. Security Considerations

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

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

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

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

The KDC should renew the group keying material after rebooting, even in the case where all keying material is stored in persistent storage. However, if the KDC relies on Observe responses to notify the group of renewed keying material, after rebooting the KDC will have lost all the current ongoing Observations with the group members, and the previous keying material will be used to protect messages in the group anyway. The KDC will rely on each node

requesting updates of the group keying material to establish the new keying material in the nodes, or, if implemented, it can push the update to the nodes in the group using the 'control_uri' resource.

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

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

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

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

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

9.1. Update of Keying Material

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

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

this by sending messages protected with the old retained keying material. Therefore, a conservative application policy should not admit the storage of old keying material.

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

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

9.2. Block-Wise Considerations

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

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

10. IANA Considerations

This document has the following actions for IANA.

10.1. 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 9 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

10.2. CoAP Content-Formats Registry

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

Media Type: application/ace-groupcomm+cbor

Encoding: -

ID: TBD

Reference: [this document]

10.3. OAuth Parameters Registry

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

* Parameter name: 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]]

10.4. OAuth Parameters CBOR Mappings Registry

The following registrations are done for the OAuth Parameters CBOR Mappings Registry following the procedure specified in Section 8.10 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]]

10.5. ACE Groupcomm Parameters Registry

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 10.14.

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

10.6. ACE Groupcomm Key Registry

This specification establishes the "ACE Groupcomm Key" IANA Registry. The Registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 10.14.

The columns of this Registry are:

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

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

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

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

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

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

10.7. ACE Groupcomm Profile Registry

This specification establishes the "ACE Groupcomm Profile" IANA Registry. The Registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 10.14. 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.

10.8. ACE Groupcomm Policy Registry

This specification establishes the "ACE Groupcomm Policy" IANA Registry. The Registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 10.14. 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 9.

10.9. Sequence Number Synchronization Method Registry

This specification establishes the "Sequence Number Synchronization Method" IANA Registry. The Registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 10.14. 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.

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

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

* Attribute Value: ace.group

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

* Reference: [This Document]

10.11. CBOR Tags Registry

This specification registers the following entry to 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]

10.12. 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 10.14. 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.

10.13. 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 10.14. 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 8. The Reference column for all of these entries refers to this document.

10.14. 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.

11. References

11.1. 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-43, 10 July 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-43.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-12, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-12.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>.

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-12, 11 May 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-mqtt-tls-profile-12.txt>.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <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>.

[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-04, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-04.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.

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

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

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

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

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

* REQ6: Specify the acceptable formats for encoding public keys and, if used, the acceptable values for 'pub_key_enc' (see Section 3.3).

* REQ7: 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).

* REQ8: Define what operations (e.g., CoAP methods) are allowed on each resource, for each role defined in REQ2 (see Section 3.3).

* REQ9: Specify the exact encoding of group identifier (see Section 4.1.1.1).

* REQ10: Specify the exact format of the 'key' value (see Section 4.1.2.1).

* REQ11: Specify the acceptable values of 'gkty' (see Section 4.1.2.1).

* REQ12: Specify the format of the identifiers of group members (see Section 4.1.2.1).

* REQ13: Specify the communication protocol the members of the group must use (e.g., multicast CoAP).

* REQ14: 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.

* REQ15: Specify and register the application profile identifier (see Section 4.1.2.1).

* REQ16: Specify policies at the KDC to handle ids that are not included in 'get_pub_keys' (see Section 4.1.3.1).

* REQ17: If used, specify the format and content of 'group_policies' and its entries. Specify the policies default values (see Section 4.1.2.1).

* REQ18: Specify the format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label (see Section 4.1.6.2).

* REQ19: Specify how the communication is secured between Client and KDC. Optionally, specify tranport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and KDC (see Section 4.3.

* REQ20: Specify the exact 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 4.1.2.1).

* REQ21: Specify how the nonce N_S is generated, if the token was not posted (e.g., if it is used directly to validate TLS instead).

* REQ22: Specify if 'mgt_key_material' used, and if yes specify its format and content (see Section 4.1.2.1). If the usage of 'mgt_key_material' is indicated and its format defined for a specific key management scheme, that format must explicitly indicate the key management scheme itself. If a new rekeying scheme is defined to be used for an existing 'mgt_key_material' in an existing profile, then that profile will have to be updated accordingly, especially with respect to the usage of 'mgt_key_material' related format and content.

* REQ23: Define the initial value of the 'num' parameter (see Section 4.1.2.1).

* REQ24: Specify and register the identifier of newly defined semantics for binary scopes (see Section 6).

* OPT1: Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' is not used (see Section 3.3).

* OPT2: Optionally, specify the additional parameters used in the Token Post exchange (see Section 3.3).

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

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

* OPT5: Optionally, specify possible or required payload formats for specific error cases.

* OPT6: Optionally, specify the behavior of the handler in case of failure to retrieve a public key for the specific node (see Section 4.1.2.1).

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

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

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

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

* OPT11: Optionally, specify additional identifiers of error types, as values of the 'error' field in an error response from the KDC.

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 Post 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 30: 'sign_info_entry' with general format

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. 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 OPT4.

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

* Possible group rekeying following an individual Key Renewal Request.

* Clarified meaning of requirement REQ3; added requirement OPT8.

* Editorial improvements.

C.2. Version -03 to -04

* Revised RESTful interface, as to methods and parameters.

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

* Revised and extended profile requirements.

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

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

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

* Editorial improvements.

C.3. 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.4. Version -01 to -02

* Editorial fixes.

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

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

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

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

* Encoding of 'pub_keys_repos' (Section 4.1).

* Encoding of 'mgt_key_material' (Section 4.1).

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

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

* Extended security considerations (Sections 10.1 and 10.2).

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

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

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

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

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

C.5. Version -00 to -01

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

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

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

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

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

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

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

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

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

* Extended security considerations (Section 10).

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

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

Acknowledgments

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

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

Authors' Addresses

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

Email: francesca.palombini@ericsson.com

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

Email: marco.tiloca@ri.se