
Stream: Internet Engineering Task Force (IETF)
RFC: [9594](#)
Category: Standards Track
Published: September 2024
ISSN: 2070-1721
Authors: F. Palombini M. Tiloca
Ericsson AB RISE AB

RFC 9594

Key Provisioning for Group Communication Using Authentication and Authorization for Constrained Environments (ACE)

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <https://www.rfc-editor.org/info/rfc9594>.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	5
1.1. Terminology	6
2. Overview	8
3. Authorization to Join a Group	11
3.1. Authorization Request	12
3.2. Authorization Response	14
3.3. Token Transferring	15
3.3.1. 'sign_info' Parameter	16
3.3.2. 'kdcchallenge' Parameter	18
4. KDC Functionalities	18
4.1. Interface at the KDC	19
4.1.1. Operations Supported by Clients	22
4.1.2. Error Handling	23
4.2. /ace-group	25
4.2.1. FETCH Handler	25
4.2.1.1. Retrieve Group Names	26
4.3. /ace-group/GROUPNAME	27
4.3.1. POST Handler	27
4.3.1.1. Join the Group	38

4.3.2. GET Handler	40
4.3.2.1. Retrieve Group Keying Material	41
4.4. /ace-group/GROUPNAME/creds	42
4.4.1. FETCH Handler	42
4.4.1.1. Retrieve a Subset of Authentication Credentials in the Group	44
4.4.2. GET Handler	45
4.4.2.1. Retrieve All Authentication Credentials in the Group	46
4.5. /ace-group/GROUPNAME/kdc-cred	47
4.5.1. GET Handler	47
4.5.1.1. Retrieve the KDC's Authentication Credential	49
4.6. /ace-group/GROUPNAME/policies	50
4.6.1. GET Handler	50
4.6.1.1. Retrieve the Group Policies	51
4.7. /ace-group/GROUPNAME/num	51
4.7.1. GET Handler	51
4.7.1.1. Retrieve the Keying Material Version	52
4.8. /ace-group/GROUPNAME/nodes/NODENAME	53
4.8.1. GET Handler	53
4.8.1.1. Retrieve Group and Individual Keying Material	54
4.8.2. POST Handler	56
4.8.2.1. Request to Change Individual Keying Material	57
4.8.3. DELETE Handler	58
4.8.3.1. Leave the Group	59
4.9. /ace-group/GROUPNAME/nodes/NODENAME/cred	59
4.9.1. POST Handler	59
4.9.1.1. Uploading an Authentication Credential	61
5. Removal of a Group Member	62
6. Group Rekeying Process	63
6.1. Point-to-Point Group Rekeying	65

6.2. One-to-Many Group Rekeying	66
6.2.1. Protection of Rekeying Messages	70
6.3. Misalignment of Group Keying Material	72
7. Extended Scope Format	73
8. ACE Groupcomm Parameters	75
9. ACE Groupcomm Error Identifiers	78
10. Security Considerations	79
10.1. Secure Communication in the Group	80
10.2. Update of Group Keying Material	80
10.3. Block-Wise Considerations	82
11. IANA Considerations	82
11.1. Media Type Registrations	82
11.2. CoAP Content-Formats	83
11.3. OAuth Parameters	83
11.4. OAuth Parameters CBOR Mappings	84
11.5. Interface Description (if=) Link Target Attribute Values	84
11.6. Custom Problem Detail Keys Registry	84
11.7. ACE Groupcomm Parameters	85
11.8. ACE Groupcomm Key Types	85
11.9. ACE Groupcomm Profiles	86
11.10. ACE Groupcomm Policies	87
11.11. Sequence Number Synchronization Methods	87
11.12. ACE Groupcomm Errors	88
11.13. ACE Groupcomm Rekeying Schemes	88
11.14. Expert Review Instructions	89
12. References	89
12.1. Normative References	89
12.2. Informative References	91
Appendix A. Requirements for Application Profiles	93
A.1. Mandatory-to-Address Requirements	93

A.2. Optional-to-Address Requirements	95
Appendix B. Extensibility for Future COSE Algorithms	96
B.1. Format of 'sign_info_entry'	96
Acknowledgments	97
Authors' Addresses	97

1. Introduction

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

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

In particular, this document defines the operations and interface available at the KDC, as well as general message formats for the interactions between Clients and the KDC. At the same time, communications in the group can rely on different approaches, e.g., based on multicast [[GROUP-CoAP](#)] or publish-subscribe (pub-sub) messaging [[CoAP-PUBSUB](#)], and can be protected in different ways.

Therefore, this document delegates details on the communication and security approaches used in a group to separate application profiles. These are specialized instances of this document that target a particular group communication approach and define how communications in the group are protected, as well as the specific keying material and configuration parameters provided to group members.

In order to ensure consistency and aid the development of such application profiles, [Appendix A](#) of this document defines a number of related compliance requirements. In particular, [Appendix A.1](#) compiles the requirements that application profiles are **REQUIRED** to fulfill; these are referred to by an identifier that starts with "REQ". Instead, [Appendix A.2](#) compiles the requirements that application profiles **MAY** fulfill; these are referred to by an identifier that starts with "OPT".

New keying material is intended to be generated and distributed to the group upon membership changes (rekeying). If the application requires backward security (i.e., new group members must be prevented from accessing communications in the group prior to their joining), then a rekeying has to occur every time new members join the group. If the application requires forward

security (i.e., former group members must be prevented from accessing communications in the group after their leaving), then a rekeying has to occur every time current members leave or are evicted from the group.

A group rekeying scheme performs the actual distribution of the new keying material by rekeying the current group members when a new Client joins the group and rekeying the remaining group members when a Client leaves the group. This can rely on different approaches, including efficient group rekeying schemes such as those described in [RFC2093], [RFC2094], and [RFC2627].

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

1.1. Terminology

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

Readers are expected to be familiar with the following:

- The terms and concepts described in the ACE framework [RFC9200] and in the Authorization Information Format (AIF) [RFC9237] to express authorization information. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).
- The terms and concepts described in the Constrained Application Protocol (CoAP) [RFC7252]. The term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".
- The terms and concepts described in Concise Data Definition Language (CDDL) [RFC8610], CBOR [RFC8949], and CBOR Object Signing and Encryption (COSE) [RFC9052] [RFC9053] [RFC9338].

A node interested in participating in group communication, as well as one that is already participating as a group member, is interchangeably denoted as "Client".

This document also uses the following terms.

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

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

This term is also not to be confused with a "multicast group", which has relevance at the network level and whose members all listen to a group network address for receiving messages sent to that group. An example of a multicast group is the set of nodes that are configured to receive messages that are sent to the group's associated IP multicast address.

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

Key Distribution Center (KDC): The entity responsible for managing one or multiple groups, with particular reference to the group membership and the keying material to use for protecting group communications.

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

Group name: The identifier of a group as a text string encoded as UTF-8 [RFC3629]. Once established, it is invariant. It is used in the interactions between the Client, AS, and RS to identify a group. A group name is always unique among the group names of the existing groups under the same KDC.

GROUPNAME: The text string used in URIs to identify a group. Once established, it is invariant. GROUPNAME uniquely maps to the group name of a group, although they do not necessarily coincide.

Group identifier: The identifier of the group keying material used in a group. Unlike group name and GROUPNAME, this identifier changes over time when the group keying material is updated.

Node name: The identifier of a node as a text string encoded as UTF-8 [RFC3629] and consistent with the semantics of URI path segments (see Section 3.3 of [RFC3986]). Once established, it is invariant. It is used in the interactions between the Client and RS, as well as to identify a member of a group. A node name is always unique among the node names of the current nodes within a group.

NODENAME: The text string used in URIs to identify a member of a group. Once established, it is invariant. Its value coincides with the node name of the associated group member.

This document additionally uses the following terminology:

Transport profile:

A profile of the ACE framework as per [Section 5.8.4.3](#) of [\[RFC9200\]](#). 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. Transport profiles of ACE include, for instance, those described in [\[RFC9202\]](#), [\[RFC9203\]](#), and [\[RFC9431\]](#).

Application profile: A 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.

Authentication credential: The set of information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [\[RFC8392\]](#), X.509 certificates [\[RFC5280\]](#), and C509 certificates [\[C509-CERT\]](#).

Individual keying material: Information pertaining exclusively to a group member, as associated with its group membership and related to other keying material and parameters used in the group. For example, this can be an identifier that the secure communication protocol employs to uniquely identify a node as a group member (e.g., a cryptographic key identifier uniquely associated with the group member in question). The specific nature and format of individual keying material used in a group is defined in the application profiles of this specification. The individual keying material of a group member is not related to the secure association between that group member and the KDC.

Throughout this document, examples for CBOR data items are expressed in CBOR extended diagnostic notation as defined in [Section 8](#) of [\[RFC8949\]](#) and [Appendix G](#) of [\[RFC8610\]](#) ("diagnostic notation"), unless noted otherwise. We often use diagnostic notation comments to provide a textual representation of the parameters' keys and values.

2. Overview

At a high level, the key provisioning process is separated in two phases: the first one follows the ACE framework between the Client, AS, and KDC, while the second one is the actual key distribution between the Client and KDC. After the two phases are completed, the Client is able to participate in the group communication via a Dispatcher entity.

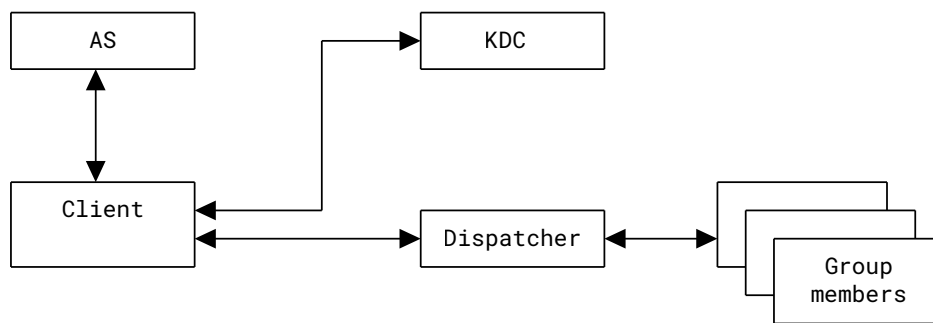


Figure 1: Key Distribution Participants

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

- Client (C): A node that wants to join a group and take part in group communication with other group members. Within the group, the Client can have different roles.
- Authorization Server (AS): As per the AS defined in the ACE framework [[RFC9200](#)], it enforces access policies that prescribe whether a node is allowed to join a given group or not and with what roles and rights (e.g., write and/or read).
- Key Distribution Center (KDC): An entity that maintains the keying material to protect group communications and provides it to Clients authorized to join a given group. During the first phase of the process ([Section 3](#)), the KDC takes the role of the RS in the ACE framework. During the second phase of the process ([Section 4](#)), which is not based on the ACE framework, the KDC distributes the keying material. In addition, the KDC provides the latest keying material to group members when requested or, if required by the application, when group membership changes.
- Group members: Nodes that have joined a group where they take part in group communication with one another, protecting it with the group keying material obtained from the KDC.
- Dispatcher: An entity through which the Clients communicate with the group when sending a message intended for multiple group members. That is, the Dispatcher distributes such a one-to-many message to the group members as intended recipients. The Dispatcher does not have access to the group keying material. A single-recipient message intended for only one group member may be delivered by alternative means, i.e., with no assistance from the Dispatcher.

Examples of a Dispatcher are: the Broker in a pub-sub setting; a relay for group communication that delivers group messages as multiple unicast messages to all group members; and an implicit entity as in a multicast communication setting, where messages are transmitted to a multicast IP address and delivered on the transport channel.

If it consists of an explicit entity, such as a pub-sub Broker or a message relay, the Dispatcher is comparable to an untrusted on-path intermediary; as such, it is able to see the messages sent by Clients in the group but not able to decrypt them and read their plain content.

This document specifies a mechanism for:

- Authorizing a Client to join the group (Section 3) and providing it with the group keying material to communicate with the other group members (Section 4),
- Allowing a group member to retrieve group keying material (Sections 4.3.2.1 and 4.8.1.1),
- Allowing a group member to retrieve authentication credentials of other group members (Section 4.4.1.1) and to provide an updated authentication credential (Section 4.9.1.1),
- Allowing a group member to leave the group (Section 4.8.3.1),
- Evicting a group member from the group (Section 5), and
- Renewing and redistributing the group keying material (rekeying), e.g., upon a membership change in the group (Section 6).

Rekeying the group may result in a temporary misalignment of the keying material stored by the different group members. Different situations where this can happen and how they can be handled are discussed in Section 6.3.

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

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

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

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

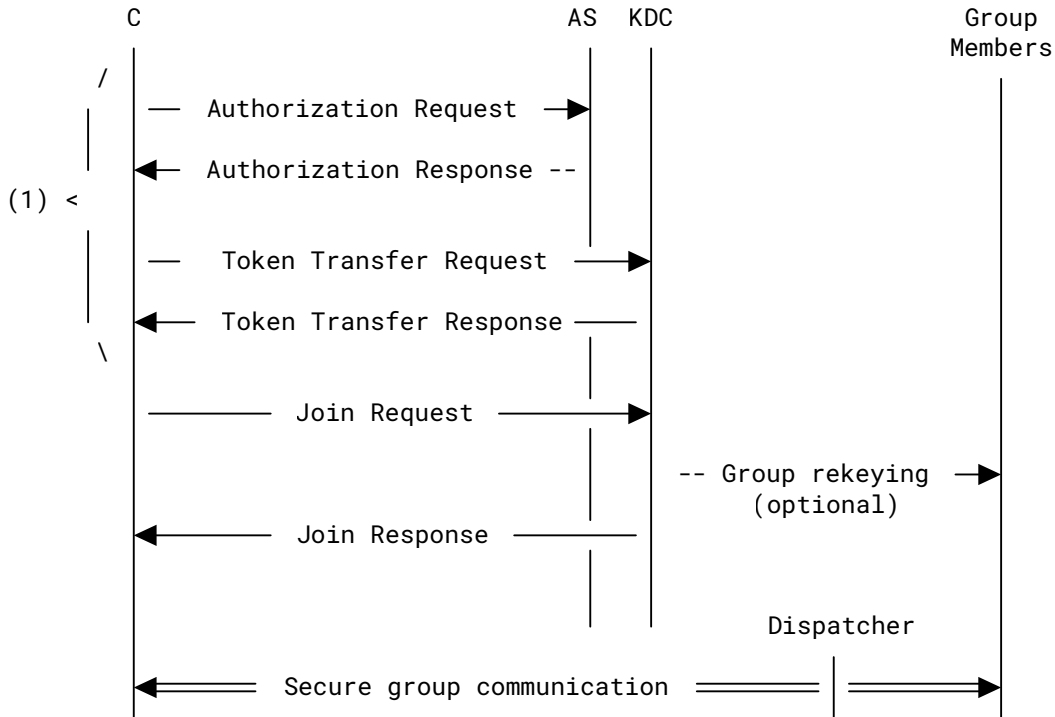
Once this exchange is completed, the joining node **MUST** have a secure communication association established with the KDC before joining a group under that KDC.

This exchange and the following secure communications between the Client and the KDC **MUST** occur in accordance with the transport profile of ACE used between the Client and KDC, such as the DTLS transport profile of ACE [RFC9202] or the OSCORE transport profile of ACE [RFC9203].

3. The joining node starts the joining process to become a member of the group by sending a request to the related group-membership resource at the KDC. Based on the application requirements and policies, the KDC may perform a group rekeying by generating new group keying material and distributing it to the current group members through the rekeying scheme used in the group.

At the end of the joining process, the joining node has received the parameters and group keying material from the KDC to securely communicate with the other group members. Also, the KDC has stored the association between the authorization information from the access token and the secure communication association with the joining node.

- 4. The joining node and the KDC maintain the secure communication association to support possible future communications. These especially include key management operations, such as the retrieval of updated keying material or the participation in a group rekeying process.
- 5. The joining node can communicate securely with the other group members by using the keying material obtained in step 3.



(1) Defined in the ACE framework

Figure 2: Message Flow upon a 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 [RFC9200].

As defined in [RFC9200], the Client asks the AS for the authorization to join the group through the KDC (see Section 3.1). If the request is approved and authorization is granted, the AS provides the Client with a proof-of-possession access token and parameters to securely communicate with the KDC (see Section 3.2).

Communications between the Client and the AS **MUST** be secured according to what is defined by the used transport profile of ACE. The Content-Format used in the message also depends on the used transport profile of ACE. For example, it can be "application/ace+cbor" for the first two messages and "application/cwt" for the third message, which are defined in the ACE framework.

The transport profile of ACE also defines a number of details, such as the communication and security protocols used with the KDC (see [Appendix C](#) of [RFC9200]).

[Figure 3](#) gives an overview of the exchange described above.

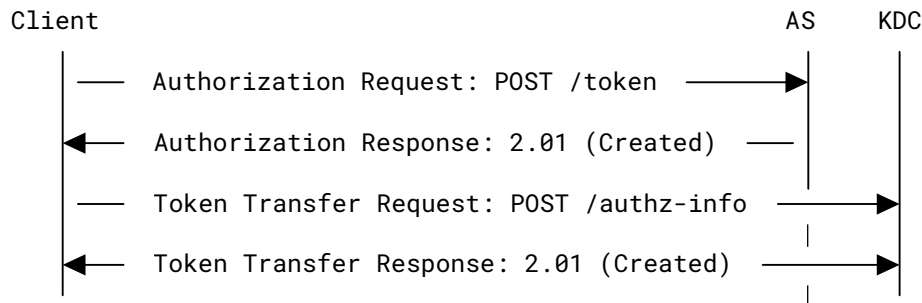


Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

The Authorization Request sent from the Client to the AS is defined in [Section 5.8.1](#) of [RFC9200] and **MAY** contain the following parameters, which, if included, **MUST** have the format and value as specified below.

- 'scope': specifying the names of the groups that the Client requests to access and optionally the roles that the Client requests to have in those groups.

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

- If AIF is used, each scope entry is encoded as per [RFC9237], i.e., as a CBOR array [Toid, Tperm]. If a scope entry expresses a set of roles to take in a group as per this document, the object identifier "Toid" specifies the group name and **MUST** be encoded as a CBOR text string, while the permission set "Tperm" specifies the roles that the Client wishes to take in the group.

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

[Figure 4](#) shows an example in CDDL notation [RFC8610] where scope uses AIF.

- If the textual format is used, each scope entry is a CBOR array formatted as follows.
 - As the first element, the group name, encoded as a CBOR text string.
 - Optionally, as the 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 with its verifiable identity credentials. Alternatively, the application may have defined a single, well-known role for the target resource(s) and audience(s).

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

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

If the application profile uses AIF, it is also **REQUIRED** to register its specific instance of "Toid" and "Tperm", as well as the corresponding media type and Content-Format, as per the guidelines in [RFC9237] (**REQ2**).

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

- 'audience': with an identifier of the KDC.

As defined in [RFC9200], other additional parameters can be included if necessary.

```

;# include rfc9237

gname = tstr

permissions = uint .bits roles

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

scope_entries = AIF-Generic<gname, permissions>

scope = bstr .cbor scope_entries

```

Figure 4: Example of scope Using AIF

```

gname = tstr

role = tstr

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

scope_entries = [* scope_entry]

scope = bstr .cbor scope_entries

```

Figure 5: Example of scope Using the Textual Format, with the Role Identifiers Encoded as Text Strings

3.2. Authorization Response

The AS processes the Authorization Request as defined in [Section 5.8.2](#) of [RFC9200], especially verifying that the Client is authorized to access the specified groups with the requested roles or possibly a subset of those.

In case of successful verification, the Authorization Response sent from the AS to the Client is also defined in [Section 5.8.2](#) of [RFC9200]. Note that the 'expires_in' parameter **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, as 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 the 'access_token' parameter **MUST** contain the following:

- a confirmation claim (for example, see 'cnf' defined in [Section 3.1](#) of [RFC8747] for CWTs)
- an expiration time claim (for example, see 'exp' defined in [Section 3.1.4](#) of [RFC8392] for CWTs)
- a scope claim (for example, see 'scope' registered in [Section 8.14](#) of [RFC9200] for CWTs)

If the 'scope' parameter is present in the Authorization Response, this claim specifies the same access control information as in the 'scope' parameter. Instead, if the 'scope' parameter is not present in the Authorization Response, this claim specifies the same access control information as in the 'scope' parameter of the Authorization Request, if the parameter is present therein, or the default scope that the AS is granting the Client otherwise.

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

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

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

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

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

When receiving an Authorization Request from a Client that was previously authorized and for which the AS still stores 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 reposting the same access token does not cause reuse of keying material between nodes (for example, that is accomplished with the use of random nonces in [RFC9203]).

3.3. Token Transferring

The Client sends a Token Transfer Request to the KDC, i.e., a CoAP POST request including the access token and targeting the /authz-info endpoint (see Section 5.10.1 of [RFC9200]).

Note that this request deviates from the one defined in [RFC9200], since it allows asking the KDC for additional information concerning the authentication credentials used in the group to ensure source authentication, as well as for possible additional group parameters.

The joining node **MAY** ask for this information from the KDC through the same Token Transfer Request. In this case, the message **MUST** have Content-Format "application/ace+cbor" registered in Section 8.16 of [RFC9200], and the message payload **MUST** be formatted as a CBOR map, which **MUST** include the access token. The CBOR map **MAY** additionally include the following parameter, which, if included, **MUST** have the format and value as specified below.

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

Alternatively, such information may be pre-configured on the joining node or may be retrieved by alternative means. For example, the joining node may have performed an early group discovery process and obtained the link to the associated group-membership resource at the KDC, along with attributes that describe the group configuration (e.g., see [OSCORE-DISCOVERY]).

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

The Token Transfer Response **MUST** have Content-Format "application/ace+cbor", and its payload is a CBOR map. Note that this deviates from what is defined in the ACE framework, where the response from the /authz-info endpoint is defined as conveying no payload (see Section 5.10.1 of [RFC9200]).

If a scope entry in the access token specifies a role that requires the Client to send its own authentication credential to the KDC when joining the related group, then the CBOR map **MUST** include the 'kdcchallenge' parameter defined in Section 3.3.2, specifying a dedicated challenge N_S generated by the KDC.

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

While storing the access token, the KDC **MUST** store the 'kdcchallenge' value associated with the Client at least until it receives a Join Request from the Client (see [Section 4.3.1.1](#)) to be able to verify the PoP evidence provided during the join process and thus that the Client possesses its own private key. The KDC deletes the 'kdcchallenge' value associated with the Client upon deleting the access token (e.g., upon its expiration, see [Section 5.10.3](#) of [\[RFC9200\]](#)).

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

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

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

Application profiles of this specification **MAY** define alternative specific negotiations of parameter values for the signature algorithm and signature keys if 'sign_info' is not used ([OPT3](#)).

If allowed by the used transport profile of ACE, the Client may provide the access token to the KDC by other means than the Token Transfer Request. An example is the DTLS transport profile of ACE, where the Client can provide the access token to the KDC during the secure session establishment (see [Section 3.3.2](#) of [\[RFC9202\]](#)).

3.3.1. 'sign_info' Parameter

The 'sign_info' parameter is an **OPTIONAL** parameter of the request and response messages exchanged between the Client and the /authz-info endpoint at the RS (see [Section 5.10.1](#) of [\[RFC9200\]](#)).

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

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

When used in the Token Transfer Request sent to the KDC (see [Section 3.3](#)), the 'sign_info' parameter specifies the CBOR simple value null (0xf6). This is done to ask for information about the signature algorithm and about the authentication credentials used in the groups that, as per the granted roles, the Client has been authorized to join or interact with (e.g., as an external signature verifier).

When used in the following Token Transfer Response from the KDC (see [Section 3.3](#)), the 'sign_info' parameter is a CBOR array of one or more elements. The number of elements is at most the number of groups that the Client has been authorized to join or interact with. Each element contains information about signing parameters and about authentication credentials for one or more groups and is formatted as follows.

- The first element 'id' is a group name or a CBOR array of group names, which is associated with groups for which the next four elements apply. Each specified group name is a CBOR text string and is hereafter referred to as 'gname'.
- The second element 'sign_alg' is a CBOR integer or a text string that indicates the signature algorithm used in the groups identified by the 'gname' values. It is **REQUIRED** for application profiles to define specific values that this parameter can take ([REQ3](#)), which are selected from the set of signing algorithms of the "COSE Algorithms" registry [[COSE.Algorithms](#)].
- The third element 'sign_parameters' is a CBOR array that indicates 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** for 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 that indicates 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** for application profiles to define the possible values and structure for the elements of this parameter ([REQ5](#)).
- The fifth element 'cred_fmt' either is a CBOR integer indicating the format of authentication credentials used in the groups identified by the 'gname' values or is the CBOR simple value null (0xf6), which indicates that the KDC does not act as a repository of authentication credentials for group members. Its acceptable integer values are taken from the "Label" column of the "COSE Header Parameters" registry [[COSE.Header.Parameters](#)], with some of those values also indicating the type of container to use for exchanging the authentication credentials with the KDC (e.g., a chain or bag of certificates). It is **REQUIRED** for application profiles to define specific values to use for this parameter, consistently with the acceptable formats of authentication credentials ([REQ6](#)).

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

```
sign_info = sign_info_req / sign_info_resp

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

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

sign_info_entry =
[
  id: gname / [+ gname],
  sign_alg: int / tstr,
  sign_parameters: [any],
  sign_key_parameters: [+ parameter: any],
  cred_fmt: int / null
]

gname = tstr
```

This format is consistent with every signature algorithm currently defined in [RFC9053], 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 response message returned from the /authz-info endpoint at the RS, as defined in Section 5.10.1 of [RFC9200]. This parameter contains a challenge generated by the RS and provided to the Client.

In this specification and in application profiles building on it, the Client can use this challenge to prove possession of its own private key in the Join Request (see the 'client_cred_verify' parameter in Section 4.3.1).

4. KDC Functionalities

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

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

A key operation that the Client can perform after transferring the access token to the KDC is a Join Request-Response exchange with the KDC. In the Join Request, the Client specifies the group it requests to join (see Section 4.3.1.1). The KDC will then check the stored access token associated with the Client and verify that the Client is accordingly authorized to join the specified group. In case of successful verification, the KDC provides the Client with the keying material to securely communicate with the other members of the group.

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

4.1. Interface at the KDC

The KDC provides its interface by hosting the following resources. Note that the root url-path "ace-group" used hereafter is a default name; implementations are not required to use this name and can define their own instead.

If request messages sent to the KDC as well as success response messages from the KDC include a payload and specify a Content-Format, those messages **MUST** have Content-Format "application/ace-groupcomm+cbor", which is registered in [Section 11.2](#). CBOR map keys used for the message parameters are defined in [Section 8](#).

- /ace-group : the path of this root resource is invariant once the resource is established. Its employment indicates that this specification is used. If other applications run on a KDC implementing this specification and use this same path, those applications will collide, and a mechanism will be needed to differentiate the endpoints.

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

Clients may be authorized to access this resource even without being members of any group managed by the KDC and even if they are not authorized to become group members (e.g., when authorized to be external signature verifiers).

The Interface Description (if=) Link Target Attribute value "ace.groups" is registered in [Section 11.5](#) and can be used to describe the interface provided by this root resource.

The example below shows an exchange with a KDC with address 2001:db8::ab that hosts the resource /ace-group and returns a link to such a resource in link-format [[RFC6690](#)].

```
Request :
Header: GET (Code=0.01)
Uri-Host: "kdc.example.com"
Uri-Path: ".well-known"
Uri-Path: "core"
Uri-Query: "if=ace.groups"

Response :
Header: Content (Code=2.05)
Content-Format: 40 (application/link-format)
Payload:
<coap://[2001:db8::ab]/ace-group>;if="ace.groups"
```

- `/ace-group/GROUPNAME` : one such sub-resource to `/ace-group` is hosted for each group with the name `GROUPNAME` that the KDC manages. In particular, it is the group-membership resource associated with that group, and it contains the symmetric group keying material of that group.

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

The Interface Description (if=) Link Target Attribute value `"ace.group"` is registered in Section 11.5 and can be used to describe the interface provided by a group-membership resource.

The example below shows an exchange with a KDC with address `2001:db8::ab` that hosts the group-membership resource `/ace-group/gp1` and returns a link to such a resource in link-format [RFC6690].

```
Request :

Header : GET (Code=0.01)
Uri-Host : "kdc.example.com"
Uri-Path : ".well-known"
Uri-Path : "core"
Uri-Query : "if=ace.group"

Response :

Header : Content (Code=2.05)
Content-Format : 40 (application/link-format)
Payload :
  <coap://[2001:db8::ab]/ace-group/gp1>;if="ace.group"
```

If it is not required that the value of the `GROUPNAME` URI path and the group name in the access token scope ('gname' in Section 3.1) coincide, the KDC **MUST** implement a mechanism to map the `GROUPNAME` value in the URI to the group name in order to refer to the correct group (REQ7).

- `/ace-group/GROUPNAME/creds` : the path of this resource is invariant once the resource is established. This resource contains the authentication credentials of all the members of the group with the name `GROUPNAME`.

This resource is created only in case the KDC acts as a repository of authentication credentials for group members.

As a group member, a Client can access this resource in order to retrieve the authentication credentials of other group members. That is, the Client can retrieve the authentication credentials of all the current group members or a subset of them by specifying filter criteria. These operations are described in Sections 4.4.2.1 and 4.4.1.1, respectively.

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

- `/ace-group/GROUPNAME/kdc-cred` : the path of this resource is invariant once the resource is established. This resource contains the authentication credential of the KDC for the group with the name `GROUPNAME`.

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

As a group member, a Client can access this resource in order to retrieve the current authentication credential of the KDC. This operation is described in [Section 4.5.1.1](#).

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

- `/ace-group/GROUPNAME/policies` : the path of this resource is invariant once the resource is established. This resource contains the group policies of the group with the name `GROUPNAME`.

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

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

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

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

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

- `/ace-group/GROUPNAME/nodes/NODENAME/cred` : the path of this resource is invariant once the resource is established. This resource contains the individual authentication credential for the node with the name `NODENAME` as a group member of the group with the name `GROUPNAME`.

A Client can access this resource in order to upload at the KDC a new authentication credential to use in the group. This operation is described in [Section 4.9.1.1](#).

This resource is not created if the group member does not have an authentication credential to use in the group or if the KDC does not store the authentication credentials of group members.

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

It is **REQUIRED** for application profiles of this specification to register a Resource Type for the group-membership resources (REQ10). This Resource Type can be used to discover the correct URL for sending a Join Request to the KDC. This Resource Type can also be used to indicate which specific application profile of this specification is used by a specific group-membership resource at the KDC.

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

4.1.1. Operations Supported by Clients

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

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

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

- GET request to /ace-group/GROUPNAME/kdc-cred in order to retrieve the current authentication credential of the KDC. This is relevant only if the KDC has an associated authentication credential and this is required for the correct group operation.
- GET request to /ace-group/GROUPNAME/policies in order to retrieve the current group policies as a group member.

- GET request to `/ace-group/GROUPNAME/nodes/NODENAME` in order to retrieve the current group keying material and individual keying material. The former can also be retrieved through a GET request to `/ace-group/GROUPNAME/` (see above).
- POST request to `/ace-group/GROUPNAME/nodes/NODENAME` in order to ask for new individual keying material. Alternatively, the Client could obtain new individual keying material by rejoining the group through a POST request to `/ace-group/GROUPNAME/` (see above). Furthermore, depending on its roles in the group or on the application profile of this specification, the Client might simply not be associated with any individual keying material.
- POST request to `/ace-group/GROUPNAME/nodes/NODENAME/cred` in order to provide the KDC with a new authentication credential. Alternatively, the Client could provide a new authentication credential by rejoining the group through a POST request to `/ace-group/GROUPNAME/` (see above). Furthermore, depending on its roles in the group, the Client might simply not have an associated authentication credential to provide.

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

4.1.2. Error Handling

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

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

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

In case the KDC stores a valid access token but the verifications above fail, the KDC **MUST** reply with a 4.03 (Forbidden) error response. This response **MAY** be an AS Request Creation Hints, as defined in [Section 5.3](#) of [\[RFC9200\]](#), in which case the Content-Format **MUST** be "application/ace+cbor".

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

If the request includes unknown or unexpected fields, the KDC **MUST** silently ignore them and continue processing the request. Application profiles of this specification **MAY** define optional or mandatory payload formats for specific error cases ([OPT4](#)).

Some error responses from the KDC can convey error-specific information according to the problem-details format defined in [RFC9290]. Such error responses **MUST** have Content-Format "application/concise-problem-details+cbor". The payload of these error responses **MUST** be a CBOR map specifying a Concise Problem Details data item (see Section 2 of [RFC9290]). The CBOR map is formatted as follows.

- It **MUST** include the Custom Problem Detail entry 'ace-groupcomm-error', which is registered in Section 11.6 of this document.

This entry is formatted as a CBOR map including only one field, namely 'error-id'. The map key for 'error-id' is the CBOR unsigned integer with value 0. The value of 'error-id' is a CBOR integer specifying the error that occurred at the KDC. This value is taken from the "Value" column of the "ACE Groupcomm Errors" registry defined in Section 11.12 of this document.

The CDDL notation [RFC8610] of the 'ace-groupcomm-error' entry is given below.

```
ace-groupcomm-error = {
    &(error-id: 0) => int
}
```

- It **MAY** include further Standard Problem Detail entries or Custom Problem Detail entries (see [RFC9290]).

In particular, it can include the Standard Problem Detail entry 'detail' (map key -2), whose value is a CBOR text string that specifies a human-readable, diagnostic description of the error occurred at the KDC. The diagnostic text is intended for software engineers as well as for device and network operators in order to aid debugging and provide context for possible intervention. The diagnostic message **SHOULD** be logged by the KDC. The 'detail' entry is unlikely relevant in an unattended setup where human intervention is not expected.

An example of an error response using the problem-details format is shown in Figure 6.

```
Response :
Header: Service Unavailable (Code=5.03)
Content-Format: 257 (application/concise-problem-details+cbor)
Payload:
{
  / title /                -1: "No available individual keying material",
  / detail /                -2: "Things will change after a
                             group rekeying; try later",
  / ace-groupcomm-error / 0: {
    / error-id / 0: 4 / "No available individual keying material" /
  }
}
```

Figure 6: Example of an Error Response with Problem Details

The problem-details format (in general) and the Custom Problem Detail entry 'ace-groupcomm-error' (in particular) are **OPTIONAL** for Clients to support. A Client supporting the entry 'ace-groupcomm-error' and that can understand the specified error may use that information to determine what actions to take next.

[Section 9](#) of this specification defines an initial set of error identifiers as possible values for the 'error-id' field. Application profiles of this specification inherit this initial set of error identifiers and **MAY** define additional values ([OPT5](#)).

4.2. /ace-group

This resource implements the FETCH handler.

4.2.1. FETCH Handler

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

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

- 'gid': its value is encoded as a CBOR array, containing one or more group identifiers. The exact encoding of the group identifier **MUST** be specified by the application profile ([REQ13](#)). The Client indicates that it wishes to receive the group names of all the groups having these identifiers.

The handler identifies the groups where communications are secured by using the keying material identified by those group identifiers.

If all verifications succeed, the handler replies with a 2.05 (Content) response, whose payload is formatted as a CBOR map that **MUST** contain the following fields:

- 'gid': its 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 for. This CBOR array is a subset of the 'gid' array in the FETCH request.
- 'gname': its 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* in this CBOR array is associated with the element of index *i* in the 'gid' array.
- 'guri': its value is encoded as a CBOR array, containing zero or more URIs, each indicating a group-membership resource. The elements of this array are encoded as text strings. Each element of index *i* in this CBOR array is associated with the element of index *i* in the 'gid' array.

If the KDC does not find any group associated with the specified group identifiers, the handler returns a response with the payload formatted as a CBOR byte string of zero length (0x40).

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

4.2.1.1. Retrieve Group Names

In case the joining node only knows the group identifier of the group it wishes to join or about which it wishes to get updated information from the KDC, the node can contact the KDC to request the corresponding group name and group-membership resource URI. In particular, it does so by sending a CoAP FETCH request to the /ace-group endpoint at the KDC formatted as defined in [Section 4.2.1](#). The node can specify several group identifiers at once.

[Figure 7](#) gives an overview of the exchanges described above, and [Figure 8](#) shows an example.

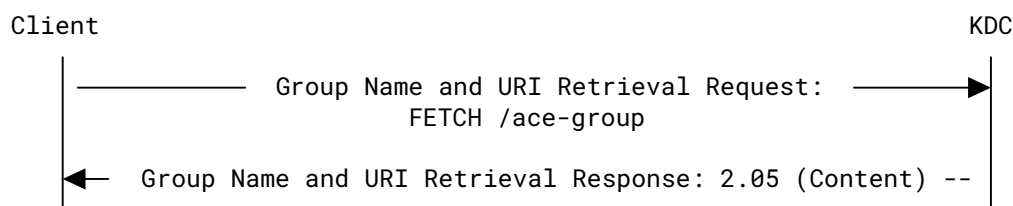


Figure 7: Message Flow of Group Name and URI Retrieval Request-Response

```

Request:
Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / gid / 0: [1, 2]
}

Response:
Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / gid / 0: [1, 2],
  / gname / 1: ["group1", "group2"],
  / guri / 2: ["/ace-group/g1", "/ace-group/g2"]
}
  
```

Figure 8: Example of Group Name and URI Retrieval Request-Response

4.3. /ace-group/GROUPNAME

This resource implements the POST and GET handlers.

4.3.1. POST Handler

The POST handler processes the Join Request sent by a Client to join a group and returns a Join Response as a successful result of the joining process (see [Section 4.3.1.1](#)). At a high level, the POST handler adds the Client to the list of current group members, adds the authentication credential of the Client to the list of the group members' authentication credentials, and returns the symmetric group keying material for the group identified by GROUPNAME.

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

- 'scope': its value specifies the name of the group that the Client is attempting to join and the roles that the client wishes to have in the group. This value is encoded as a CBOR byte string wrapping one scope entry, as defined in [Section 3.1](#).
- 'get_creds': it is included if the Client wishes to receive the authentication credentials of the current group members from the KDC. This parameter may be included in the Join Request if the KDC stores the authentication credentials of the group members, while it is not useful to include it if the Client obtains those authentication credentials through alternative means, e.g., from the AS. Note that including this parameter might result in a following Join Response of a large size, which can be inconvenient for resource-constrained devices.

If the Client wishes to retrieve the authentication credentials of all the current group members, the 'get_creds' parameter **MUST** encode the CBOR simple value `null` (0xf6). Otherwise, if the Client wishes to retrieve the authentication credentials of nodes with specific roles, the 'get_creds' parameter **MUST** encode a non-empty CBOR array containing the three elements 'inclusion_flag', 'role_filter', and 'id_filter', as defined below.

- The first element, namely 'inclusion_flag', encodes the CBOR simple value `true` (0xf5) if the Client wishes to receive the authentication credentials of the nodes having their node identifier specified in 'id_filter' (i.e., selection by inclusive filtering). Instead, this element encodes the CBOR simple value `false` (0xf4) if the Client wishes to receive the authentication credentials of the nodes that do not have the node identifiers specified in the third element 'id_filter' (i.e., selection by exclusive filtering). In the Join Request, this parameter encodes the CBOR simple value `true` (0xf5).
- The second element, namely 'role_filter', is a CBOR array. Each element of the array contains one role or a combination of roles for the group identified by GROUPNAME. This parameter indicates that the Client wishes to receive the authentication credentials of all the group members having any of the specified roles or combination of roles (i.e., having any of those single roles or at least all the roles indicated in any of those combinations of roles).

For example, the array ["role1", "role2+role3"] indicates that the Client wishes to receive the authentication credentials of all group members that have at least "role1" or at least both "role2" and "role3". In the Join Request, this parameter is a non-empty array.

- The third element, namely 'id_filter', is a CBOR array. Each element of the array contains a node identifier of a group member for the group identified by GROUPNAME. This parameter indicates that the Client wishes to receive the authentication credentials of the nodes that have or do not have the specified node identifiers based on the value of 'inclusion_flag' (i.e., as a selection by inclusive or exclusive filtering). In the Join Request, the Client does not filter authentication credentials based on node identifiers, so this parameter is an empty array.

In fact, when first joining the group, the Client is not expected or capable to express a filter based on node identifiers of other group members. Instead, when already a group member and sending a Join Request to rejoin, the Client is not expected to include the 'get_creds' parameter in the Join Request altogether, since it can rather retrieve authentication credentials associated with specific group identifiers as defined in [Section 4.4.1.1](#).

The CDDL definition [RFC8610] of 'get_creds' is given in [Figure 9](#); as an example, it uses node identifiers encoded as CBOR byte strings, role identifiers encoded as CBOR text strings, and combinations of roles encoded as CBOR arrays of role identifiers.

Note that, for this handler, 'inclusion_flag' is always set to true and the array of roles 'role_filter' is always non-empty, while the array of node identifiers 'id_filter' is always empty. However, this is not necessarily the case for other handlers using the 'get_creds' parameter.

```

inclusion_flag = bool

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

id = bstr
id_filter = [* id]

get_creds = null / [inclusion_flag, role_filter, id_filter]

```

Figure 9: CDDL Definition of 'get_creds', Using an Example Node Identifier Encoded as bstr and Role as tstr

- 'client_cred': encoded as a CBOR byte string, whose value is the original binary representation of the Client's authentication credential. This parameter **MUST** be present if the KDC is managing (collecting from and distributing to Clients) the authentication credentials of the group members and the Client's role in the group will require the Client to send messages to one or more group members. It is **REQUIRED** for application profiles to define the specific formats that are acceptable to use for authentication credentials in the group ([REQ6](#)).
- 'nonce': encoded as a CBOR byte string, whose value is a dedicated nonce N_C generated by the Client. This parameter **MUST** be present.

- 'client_cred_verify': encoded as a CBOR byte string. This parameter **MUST** be present if the 'client_cred' parameter is present and no authentication credential associated with the Client's access token can be retrieved for that group.

The value of the CBOR byte string is the proof-of-possession (PoP) evidence computed by the Client over the following PoP input: the scope (encoded as a CBOR byte string) concatenated with N_S (encoded as a CBOR byte string) concatenated with N_C (encoded as a CBOR byte string), where:

- scope is either specified in the 'scope' parameter above, if present, or a default scope entry that the handler is expected to know otherwise;
- N_S is the challenge received from the KDC in the 'kdcchallenge' parameter of the 2.01 (Created) response to the Token Transfer Request (see [Section 3.3](#)), encoded as a CBOR byte string; and
- N_C is the nonce generated by the Client and specified in the 'nonce' 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 10](#).

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 authentication credential carried in the 'client_cred' parameter. Application profiles of this specification **MUST** specify the exact approaches used to compute the PoP evidence to include in 'client_cred_verify' and **MUST** specify which of those approaches is used in which case ([REQ14](#)).

If the access token was not provided to the KDC through a Token Transfer Request (e.g., the access token is instead transferred during the establishment of a secure communication association), it is **REQUIRED** of the specific application profile to define how the challenge N_S is generated ([REQ15](#)).

- 'creds_repo': it can be present if the format of the Client's authentication credential conveyed in the 'client_cred' parameter is a certificate. In such a case, this parameter has as its 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 application profiles of this specification ([OPT6](#)).
- 'control_uri': its value is a full URI, encoded as a CBOR text string. A default url-path is /ace-group/GROUPNAME/node, although implementations can use different ones instead. The URI **MUST NOT** have url-path /ace-group/GROUPNAME.

If 'control_uri' is specified in the Join Request, the Client acts as a CoAP server and hosts a resource at this specific URI. The KDC **MAY** use this URI to send CoAP requests to the Client (acting as a CoAP server in this exchange), for example, for one-to-one provisioning of new group keying material when performing a group rekeying (see [Section 6.1](#)) or to inform the Client of its removal from the group (see [Section 5](#)).

In particular, this resource is intended for communications exclusively concerning the group identified by GROUPNAME and whose group name is specified in the 'scope' parameter of the Join Request, if present therein. If the KDC does not implement mechanisms using this resource for that group, it can ignore this parameter. Other additional functionalities of this resource **MAY** be defined in application profiles of this specifications ([OPT7](#)).

```
scope, N_S, and N_C expressed in CBOR diagnostic notation:  
  scope = h'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 10: Example of PoP Input to Compute 'client_cred_verify' Using CBOR Encoding

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

The handler considers the scope specified in the access token associated with the Client and checks the scope entry related to the group identified by the GROUPNAME associated with the endpoint. In particular, the handler checks whether the set of roles specified in that scope entry includes all the roles that the Client wishes to have in the group as per the Join Request. If this is not the case, the KDC **MUST** reply with a 4.03 (Forbidden) error response.

If the KDC manages the group members' authentication credentials, the handler checks if one is included in the 'client_cred' parameter. If so, the KDC retrieves the authentication credential and performs the following actions.

- If the access token was provided through a Token Transfer Request (see [Section 3.3](#)) but the KDC cannot retrieve the 'kdcchallenge' associated with this Client (see [Section 3.3](#)), the KDC **MUST** reply with a 4.00 (Bad Request) error response, which **MUST** also have Content-Format "application/ace-groupcomm+cbor". The payload of the error response is a CBOR map including a newly generated 'kdcchallenge' value, which is specified in the 'kdcchallenge' parameter. The KDC **MUST** store the newly generated value as the 'kdcchallenge' value associated with this Client, replacing the currently stored value (if any).

- The KDC checks the authentication credential to be valid for the group identified by GROUPNAME. That is, it checks that the authentication credential has the format used in the group, is intended for the public key algorithm used in the group, and is aligned with the possible associated parameters used in the group.

If this verification fails, the handler **MUST** reply with a 4.00 (Bad Request) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 2 ("Authentication credential incompatible with the group configuration").

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

If the PoP evidence does not pass verification, the handler **MUST** reply with a 4.00 (Bad Request) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 3 ("Invalid proof-of-possession evidence").

If no authentication credential is conveyed in the 'client_cred' parameter, the handler checks if the KDC currently stores an authentication credential that is associated with the access token and with the group identified by GROUPNAME (see also [Section 4.3.1.1](#)). Note that the same joining node may use different authentication credentials in different groups, and all those authentication credentials would be associated with the same access token.

If an eligible authentication credential for the Client is neither present in the 'client_cred' parameter nor retrieved from the stored ones at the KDC, it is **RECOMMENDED** that the handler stops the processing and replies with a 4.00 (Bad Request) error response. Application profiles **MAY** define alternatives ([OPT8](#)).

If, regardless of the reason, the KDC replies with a 4.00 (Bad Request) error response, the payload of the response **MAY** be a CBOR map. For instance, the CBOR map can include a 'sign_info' parameter formatted as 'sign_info_resp' defined in [Section 3.3.1](#), with the 'cred_fmt' element set to the CBOR simple value null (0xf6) if the Client sent its own authentication credential and the KDC is not set to store authentication credentials of the group members. When the response payload is a CBOR map including such parameters, the error response has Content-Format "application/ace-groupcomm+cbor".

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

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

- The handler adds the Client to the list of current members of the group.
- The handler assigns a name NODENAME to the Client and creates a sub-resource to /ace-group/GROUPNAME at the KDC, i.e., /ace-group/GROUPNAME/nodes/NODENAME.

- The handler associates the node identifier NODENAME with the access token and the secure communication association for the Client.

Then, the handler performs the following actions.

- If the KDC manages the group members' authentication credentials:
 - The handler associates the retrieved Client's authentication credential with the tuple composed of the node name NODENAME, the group name GROUPNAME, and the access token.
 - The handler adds the retrieved Client's authentication credential to the list of authentication credentials stored for the group identified by GROUPNAME. If such a list already includes an authentication credential for the Client, but a different authentication credential is specified in the 'client_cred' parameter, then the handler **MUST** replace the old authentication credential in the list with the one specified in the 'client_cred' parameter.
- If backward security is prescribed by application policies installed at the KDC or by the used application profile of this specification, then the KDC **MUST** generate new group keying material and securely distribute it to the current group members (see [Section 6](#)).
- The handler returns a successful Join Response, as defined below, which contains the symmetric group keying material, the group policies, and the authentication credentials of the current members of the group if the KDC manages those and the Client requested those.

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

The Join Response message **MUST** include the Location-Path CoAP Options, specifying the path to the sub-resource associated with the Client, i.e., /ace-group/GROUPNAME/nodes/NODENAME.

The Join Response message **MUST** have Content-Format "application/ace-groupcomm+cbor". The payload of the response is formatted as a CBOR map, which **MUST** contain the following fields with the values specified below:

- 'gkty': identifying the key type of the keying material specified in the 'key' parameter. This parameter is encoded as a CBOR integer or a CBOR text string. Possible values are taken from the "Key Type Value" column of the "ACE Groupcomm Key Types" registry defined in [Section 11.8](#) of this specification. Implementations **MUST** verify that the key type specified by this parameter matches the application profile being used and, if applicable, that such an application profile is listed in the "Profile" column of the "ACE Groupcomm Key Types" registry for the key type in question.
- 'key': containing the keying material used for securing the group communication or information required to derive such keying material.
- 'num': containing the current version number of the group keying material, encoded as a CBOR integer. The version number has a value that increases in a strictly monotonic way as the group keying material changes. The application profile **MUST** define the initial value of the version number ([REQ16](#)).

The format of the keying material conveyed in the 'key' parameter **MUST** be defined in application profiles of this specification (REQ17), together with corresponding key types to specify as value of the 'gkty' parameter and that are accepted by the application (REQ18). Additionally, documents specifying a type of keying material **MUST** register an entry in the "ACE Groupcomm Key Types" registry defined in Section 11.8, including its name, the corresponding key type to specify as value for the 'gkty' parameter, and the application profile to be used with.

Name	Key Type Value	Profile	Description	Reference
Reserved	0		This value is reserved	RFC 9594

Table 1: ACE Groupcomm Key Types

The Join Response **SHOULD** contain the following fields with the values specified below:

- 'exp': its value specifies the expiration time of the group keying material specified in the 'key' parameter, encoded as a CBOR unsigned integer. The value is the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds, analogous to what is specified for NumericDate in Section 2 of [RFC7519]. After the time indicated in this parameter, group members **MUST NOT** use the group keying material specified in the 'key' parameter. The group members can retrieve the latest group keying material from the KDC.
- 'exi': its value specifies the residual lifetime of the group keying material, encoded as a CBOR unsigned integer. If the 'exp' parameter is included, this parameter **MUST** also be included. The value represents the residual lifetime of the group keying material specified in the 'key' parameter, i.e., it is the number of seconds between the current time at the KDC and the time when the keying material expires (as specified in the 'exp' parameter, if present). A Client determines the expiration time of the keying material by adding the seconds specified in the 'exi' parameter to its current time upon receiving the Join Response containing the 'exi' parameter. After such an expiration time, the Client **MUST NOT** use the group keying material specified in the 'key' parameter. The Client can retrieve the latest group keying material from the KDC.

If a Client has a reliable way to synchronize its internal clock with UTC, and both the 'exp' and 'exi' parameters are present, then the Client **MUST** use the 'exp' parameter value as expiration time for the group keying material. Otherwise, the Client uses the 'exi' parameter value to determine the expiration time as defined above.

When a Client relies on the 'exi' parameter, the expiration time that it computes is offset in the future with respect to the actual expiration time as intended by the KDC and specified in the 'exp' parameter (if present). Such an offset is the amount of time between when the KDC sends the response message including the 'exi' parameter and when the Client receives that message. That is, especially if the delivery of the response to the Client is delayed, the Client will believe the keying material to be valid for a longer time than the KDC actually means. However, before approaching the actual expiration time, the KDC is expected to rekey the group and distribute new keying material (see Section 6).

Optionally, the Join Response **MAY** contain the following parameters, which, if included, **MUST** have the format and value as specified below.

- 'ace_groupcomm_profile': its value is encoded as a CBOR integer and **MUST** be used to uniquely identify the application profile for group communication. Applications of this specification **MUST** register an application profile identifier and the related value for this parameter in the "ACE Groupcomm Profiles" registry ([REQ19](#)).

Name	Description	CBOR Value	Reference
Reserved	This value is reserved	0	RFC 9594

Table 2: ACE Groupcomm Profiles

- 'creds': it **MUST** be present if 'get_creds' was present in the Join Request; otherwise, it **MUST NOT** be present. Its value is encoded as a CBOR array specifying the authentication credentials of the group members, i.e., of all of them or of the ones selected according to the 'get_creds' parameter in the Join Request. In particular, each element of the array is a CBOR byte string, whose value is the original binary representation of a group member's authentication credential. It is **REQUIRED** for application profiles to define the specific formats of authentication credentials that are acceptable to use in the group ([REQ6](#)).
- 'peer_roles': it **SHOULD** be present if 'creds' is also present; otherwise, it **MUST NOT** be present. Its value is encoded as a CBOR array of n elements, where n is the number of authentication credentials included in the 'creds' parameter (at most, the number of members in the group). The i-th element of the array specifies the role(s) that the group member associated with the i-th authentication credential in 'creds' has in the group. In particular, each array element is encoded like the role element of a scope entry, which is consistent with the used format (see [Section 3.1](#)).

This parameter **MAY** be omitted if the Client can rely on other means to unambiguously gain knowledge of the role of each group member whose associated authentication credential is specified in the 'creds' parameter. For example, all such group members may have the same role in the group joined by the Client, and such a role can be unambiguously assumed by the Client (e.g., based on what is defined in the used application profile of this specification). As another example, each of the authentication credentials specified in the 'creds' parameter can indicate the role(s) that the corresponding group member has in the group joined by the Client.

When receiving the authentication credential of a Client in the 'client_cred' parameter of a Join Request (see [Section 4.3.1.1](#)) or of an Authentication Credential Update Request (see [Section 4.9.1.1](#)), the KDC is not expected to check that the authentication credential indicates the role(s) that the Client can have or has in the group in question. When preparing a Join Response, the KDC can decide whether to include the 'peer_roles' parameter, depending on the specific set of authentication credentials specified in the 'creds' parameter of that Join Response.

- 'peer_identifiers': it **MUST** be present if 'creds' is also present; otherwise, it **MUST NOT** be present. Its value is encoded as a CBOR array of n elements, where n is the number of authentication credentials included in the 'creds' 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 with the i -th authentication credential in 'creds' has in the group. In particular, the i -th array element is encoded as a CBOR byte string, whose value is the node identifier of the group member. The specific format of node identifiers of group members is specified by the application profile (REQ25).

- 'group_policies': its value is encoded as a CBOR map, whose elements specify how the group handles specific management aspects. These include, for instance, approaches to achieve synchronization of sequence numbers among group members. The possible elements of the CBOR map are registered in the "ACE Groupcomm Policies" registry defined in Section 11.10 of this specification. This specification defines the three elements "Sequence Number Synchronization Methods", "Key Update Check Interval", and "Expiration Delta", which are summarized in Table 3. Application profiles of this specification **MUST** specify the format and default values for the entries of the CBOR map conveyed in the 'group_policies' parameter (REQ20).

Name	CBOR Label	CBOR Type	Description	Reference
Sequence Number Synchronization Method	0	int or tstr	Method for recipient group members to synchronize with sequence numbers of sender group members. Its value is taken from the "Value" column of the "Sequence Number Synchronization Method" registry.	RFC 9594
Key Update Check Interval	1	int	Polling interval in seconds, for group members to check at the KDC if the latest group keying material is the one that they store.	RFC 9594
Expiration Delta	2	uint	Number of seconds from 'exp' until a UTC date/time, after which group members MUST stop using the group keying material that they store to decrypt incoming messages.	RFC 9594

Table 3: ACE Groupcomm Policies

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

If the KDC has an authentication credential as required for the correct group operation, the KDC's authentication credential **MUST** have the same format used for the authentication credentials of the group members. It is **REQUIRED** for application profiles to define the specific formats that are acceptable to use for the authentication credentials in the group (REQ6).

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

The value of this parameter is the proof-of-possession (PoP) evidence computed by the KDC over the following PoP input: the nonce N_C (encoded as a CBOR byte string) concatenated with the nonce N_KDC (encoded as a CBOR byte string), where:

- N_C is the nonce generated by the Client and specified in the 'cnonce' parameter of the Join Request.
- N_KDC is the nonce generated by the KDC and specified in the 'kdc_nonce' parameter.

An example of PoP input to compute 'kdc_cred_verify' using CBOR encoding is given in Figure 11.

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

- 'rekeying_scheme': identifying the rekeying scheme that the KDC uses to provide new group keying material to the group members. The value of this parameter is encoded as a CBOR integer and is taken from the "Value" column of the "ACE Groupcomm Rekeying Schemes" registry defined in Section 11.13 of this specification.

Value	Name	Description	Reference
0	Point-to-Point	The KDC individually targets each node to rekey, using the pairwise secure communication association with that node	RFC 9594

Table 4: ACE Groupcomm Rekeying Schemes

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

- 'mgt_key_material': encoded as a CBOR byte string and containing the specific administrative keying material that the joining node requires in order to participate in the group rekeying process performed by the KDC. This parameter **MUST NOT** be present if the 'rekeying_scheme' parameter is not present and the application profile does not specify a default group

rekeying scheme to use in the group. Some simple rekeying schemes may not require specific administrative keying material to be provided, e.g., the basic "Point-to-Point" group rekeying scheme (see [Section 6.1](#)).

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

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

- 'control_group_uri': its value is a full URI, encoded as a CBOR text string. The URI **MUST** specify addressing information intended to reach all the members in the group. For example, this can be a multicast IP address, optionally together with a port number that, if omitted, defaults to 5683, i.e., the default port number for the "coap" URI scheme (see [Section 6.1](#) of [\[RFC7252\]](#)). The URI **MUST** include GROUPNAME in the url-path. A default url-path is /ace-group/GROUPNAME, although implementations can use different ones instead. The URI **MUST NOT** have url-path /ace-group/GROUPNAME/nodes.

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

The KDC **MAY** use this URI to send one-to-many CoAP requests to the Client group members (acting as CoAP servers in this exchange), for example, for one-to-many provisioning of new group keying material when performing a group rekeying (see [Section 6.2](#)) or to inform the Clients of their removal from the group (see [Section 5](#)).

In particular, this resource is intended for communications exclusively concerning the group identified by GROUPNAME and whose group name was specified in the 'scope' parameter of the Join Request, if present. If the KDC does not implement mechanisms using this resource for that group, it can ignore this parameter. Other additional functionalities of this resource **MAY** be defined in application profiles of this specifications ([OPT10](#)).

N_C and N_KDC expressed in CBOR diagnostic notation:

```
N_C   = h'25a8991cd700ac01'
N_KDC = h'cef04b2aa791bc6d'
```

N_C and N_KDC as CBOR encoded byte strings:

```
N_C   = 0x4825a8991cd700ac01
N_KDC = 0x48cef04b2aa791bc6d
```

PoP input:

```
0x48 25a8991cd700ac01 48 cef04b2aa791bc6d
```

Figure 11: Example of PoP Input to Compute 'kdc_cred_verify' Using CBOR Encoding

After sending the Join Response, if the KDC has an associated authentication credential as required for the correct group operation, then the KDC **MUST** store the N_C value specified in the 'cnonce' parameter of the Join Request as a 'clientchallenge' value associated with the Client, replacing the currently stored value (if any). If, as a group member, the Client later sends a GET request to the /ace-group/GROUPNAME/kdc-cred resource for retrieving the latest KDC's authentication credential (see [Section 4.5.1](#)), then the KDC uses the stored 'clientchallenge' for computing the PoP evidence to include in the response sent to the Client, hence proving the possession of its own private key.

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

Application profiles of this specification **MUST** specify the communication protocol that members of the group use to communicate with each other ([REQ22](#)) and the security protocol that they use to protect the group communication ([REQ23](#)).

4.3.1.1. Join the Group

[Figure 12](#) gives an overview of the join exchange between the Client and the KDC when the Client first joins a group, while [Figure 13](#) shows an example.

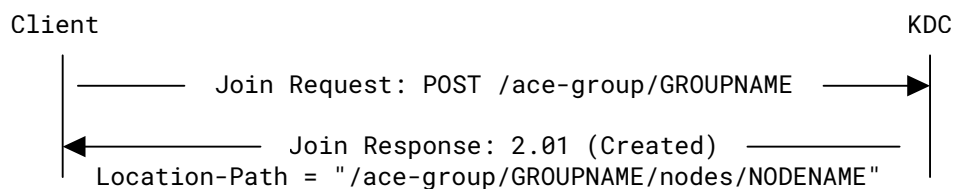


Figure 12: Message Flow of the Join Request-Response

Request:

```

Header: POST (Code=0.02)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / scope /                3: <<["group1", ["sender", "receiver"]]>>,
  / get_creds /            4: [true, ["sender"], []],
  / client_cred /         5: h'a2026008a101a5010202410a20012158
                           20bbc34960526ea4d32e940cad2a2341
                           48ddc21791a12afbcbac93622046dd44
                           f02258204519e257236b2a0ce2023f09
                           31f1f386ca7afda64fcde0108c224c51
                           eabf6072',
  / cnonce /              6: h'25a8991cd700ac01',
  / client_cred_verify / 24: h'66e6d9b0db009f3e105a673f88556117
                           26caed57f530f8cae9d0b168513ab949
                           fedc3e80a96ebe94ba08d3f8d3bf8348
                           7458e2ab4c2f936ff78b50e33c885e35'
}

```

Response:

```

Header: Created (Code=2.01)
Content-Format: 261 (application/ace-groupcomm+cbor)
Location-Path: "ace-group"
Location-Path: "g1"
Location-Path: "nodes"
Location-Path: "c101"
Payload (in CBOR diagnostic notation):
{
  / gkty /                 7: 65600,
  / key /                 8: h'73657373696f6e6b6579',
  / num /                 9: 12,
  / exp /                11: 1924992000,
  / exi /                12: 2592000,
  / creds /              13: [h'a2026008a101a5010202410220012158
                           20cd4177ba62433375ede279b5e18e8b
                           91bc3ed8f1e174474a26fc0edb44ea53
                           73225820a0391de29c5c5badda610d4e
                           301eaaa18422367722289cd18cbe6624
                           e89b9cfd',
                           h'a2026008a101a5010202410320012158
                           20ac75e9ece3e50bfc8ed60399889522
                           405c47bf16df96660a41298cb4307f7e
                           b62258206e5de611388a4b8a8211334a
                           c7d37ecb52a387d257e6db3c2a93df21
                           ff3affc8'],
  / peer_roles /         14: ["sender", ["sender", "receiver"]],
  / peer_identifiers / 15: [h'01', h'02']
}

```


Figure 13: Example of the First Join Request-Response for Group Joining

If not previously established, the Client and the KDC **MUST** first establish a pairwise secure communication association (REQ24). This can be achieved, for instance, by using a transport profile of ACE. The join exchange **MUST** occur over that secure communication association. The Client and the KDC **MAY** use that same secure communication association to protect further pairwise communications that must be protected.

It is **REQUIRED** that the secure communication association between the Client and the KDC is established 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 the pairwise secure communication association 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 the group to join is identified by GROUPNAME. The group name is specified in the scope entry conveyed by the 'scope' parameter of the request (if present), formatted as specified in Section 4.3.1. This group name is the same as in the scope entry corresponding to that group, specified in the 'scope' parameter of the Authorization Request/Response, or it can be determined from it. Note that, in case of successful joining, the Location-Path Options in the Join Response provide the Client with the path of the URI to use for retrieving individual keying material and for leaving the group.

If the node is joining a group for the first time and the KDC maintains the authentication credentials of the group members, the Client is **REQUIRED** to send its own authentication credential and proof-of-possession (PoP) evidence in the Join Request (see the 'client_cred' and 'client_cred_verify' parameters in Section 4.3.1). The request is accepted only if both the authentication credential is provided and the PoP evidence is successfully verified.

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

4.3.2. GET Handler

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

The handler expects a GET request.

In addition to what is defined in Section 4.1.2, the handler verifies that the Client is a current member of the group. If the verification fails, the KDC **MUST** reply with a 4.03 (Forbidden) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor"

and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 0 ("Operation permitted only to group members").

If all verifications succeed, the handler replies with a 2.05 (Content) response containing the symmetric group keying material. The payload of the response is formatted as a CBOR map that **MUST** contain the parameters 'gkty', 'key', and 'num', as specified in [Section 4.3.1](#).

The payload **MUST** also include the parameters 'rekeying_scheme' and 'mgt_key_material' as specified in [Section 4.3.1](#), if they are included in the payload of the Join Responses sent for the group.

The payload **MAY** also include the parameters 'ace_groupcomm_profile', 'exp', and 'exi', as specified in [Section 4.3.1](#). If the 'exp' parameter is included, the 'exi' parameter **MUST** also be included. If the 'exi' parameter is included, its value specifies the residual lifetime of the group keying material from the current time at the KDC.

4.3.2.1. Retrieve Group Keying Material

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

[Figure 14](#) gives an overview of the key distribution exchange between the Client and the KDC, while [Figure 15](#) shows an example.

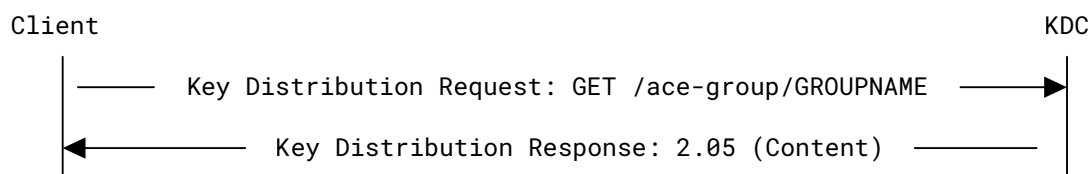


Figure 14: Message Flow of Key Distribution Request-Response

```
Request :

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

Response :

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / gkty / 7: 65600,
  / key / 8: h'7365737369666e6b6579',
  / num / 9: 12
}
```

Figure 15: Example of Key Distribution Request-Response

4.4. /ace-group/GROUPNAME/creds

This resource implements the GET and FETCH handlers.

4.4.1. FETCH Handler

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

The handler expects a request with the payload formatted as a CBOR map, which **MUST** contain the following field.

- 'get_creds': its value is encoded as in [Section 4.3.1](#), with the following modifications.
 - The arrays 'role_filter' and 'id_filter' **MUST NOT** both be empty, i.e., in CDDL notation: [bool, [], []]. If the 'get_creds' parameter has such a format, the request **MUST** be considered malformed, and the KDC **MUST** reply with a 4.00 (Bad Request) error response.
Note that a group member can retrieve the authentication credentials of all the current group members by sending a GET request to the same KDC resource instead (see [Section 4.4.2.1](#)).
 - The element 'inclusion_flag' encodes the CBOR simple value true (0xf5) or false (0xf4), as defined in [Section 4.3.1](#).
 - The array 'role_filter' can be empty if the Client does not wish to filter the requested authentication credentials 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, as defined in [Section 4.3.1](#). The array may be empty if the Client does not wish to filter the requested authentication credentials based on the node identifiers of the group members.

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

- If the 'inclusion_flag' encodes the CBOR simple value `true` (0xf5), the handler returns the authentication credentials of group members whose roles match with 'role_filter' and/or have their node identifier specified in 'id_filter'.
- If the 'inclusion_flag' encodes the CBOR simple value `false` (0xf4), the handler returns the authentication credentials of group members whose roles match with 'role_filter' and, at the same time, do not have their node identifier specified in 'id_filter'.

The specific format of authentication credentials as well as the identifiers, roles, and combination of roles of group members **MUST** be specified by application profiles of this specification ([REQ1](#), [REQ6](#), [REQ25](#)).

The handler identifies the authentication credentials of the current group members for which either of the following holds:

- The role identifier matches with one of those indicated in the request; note that the request can specify a combination of roles, in which case the handler selects only the group members that have all the roles included in the combination.
- The node identifier matches with one of those indicated in the request or does not match with any of those, which is consistent with the value of the element 'inclusion_flag'.

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

- 'num': encoding the version number of the current group keying material.
- 'creds': encoding the list of authentication credentials of the selected group members.
- 'peer_roles': encoding the role(s) that each of the selected group members has in the group. This parameter **SHOULD** be present, and it **MAY** be omitted according to the same criteria defined for the Join Response (see [Section 4.3.1](#)).
- 'peer_identifiers': encoding the node identifier that each of the selected group members has in the group.

The specific format of authentication credentials as well as of node identifiers of group members is specified by the application profile ([REQ6](#), [REQ25](#)).

If the KDC does not store any authentication credential associated with the specified node identifiers, the handler returns a response with the payload formatted as a CBOR byte string of zero length (0x40).

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_creds' parameter of the request but are not associated with any current group member. Such a policy **MUST** be specified by application profiles of this specification ([REQ26](#)).

- The KDC silently ignores those node identifiers.

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

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

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

4.4.1.1. Retrieve a Subset of Authentication Credentials in the Group

In case the KDC maintains the authentication credentials of group members, a node in the group can contact the KDC to request the authentication credentials, roles, and node identifiers of a specified subset of group members by sending a CoAP FETCH request to the `/ace-group/GROUPNAME/creds` endpoint at the KDC, which is formatted as defined in [Section 4.4.1](#) and where GROUPNAME identifies the group.

[Figure 16](#) gives an overview of the exchange mentioned above, while [Figure 17](#) shows an example of such an exchange.

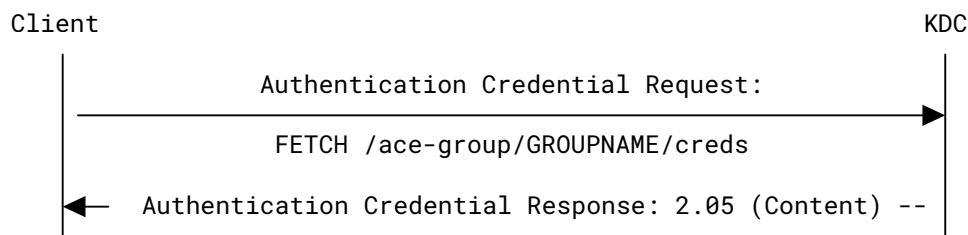


Figure 16: Message Flow of Authentication Credential Request-Response to Obtain the Authentication Credentials of Specific Group Members

```

Request:

Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "creds"
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / get_creds / 4: [true, [], [h'02', h'03']]
}

Response:

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / creds /
    13: [h' a2026008a101a5010202410320012158
          20ac75e9ece3e50bfc8ed60399889522
          405c47bf16df96660a41298cb4307f7e
          b62258206e5de611388a4b8a8211334a
          c7d37ecb52a387d257e6db3c2a93df21
          ff3affc8',
          h' a2026008a101a5010202410920012158
          206f9702a66602d78f5e81bac1e0af01
          f8b52810c502e87ebb7c926c07426fd0
          2f225820c8d33274c71c9b3ee57d842b
          bf2238b8283cb410eca216fb72a78ea7
          a870f800' ],
  / peer_roles / 14: [{"sender", "receiver"}, "receiver"],
  / peer_identifiers / 15: [h'02', h'03']
}

```

Figure 17: Example of Authentication Credential Request-Response to Obtain the Authentication Credentials of Specific Group Members

4.4.2. GET Handler

The handler expects a GET request.

If all verifications succeed, the KDC replies with a 2.05 (Content) response as in the FETCH handler in [Section 4.4.1](#), but its payload specifies the authentication credentials of all the group members, together with their roles and node identifiers.

The 'peer_roles' parameter **SHOULD** be present in the payload of the response, and it **MAY** be omitted according to the same criteria defined for the Join Response (see [Section 4.3.1](#)).

4.4.2.1. Retrieve All Authentication Credentials in the Group

In case the KDC maintains the authentication credentials of group members, a node in the group or an external signature verifier can contact the KDC to request the authentication credentials, roles, and node identifiers of all the current group members, by sending a CoAP GET request to the `/ace-group/GROUPNAME/creds` endpoint at the KDC, where the group is identified by `GROUPNAME`.

Figure 18 gives an overview of the message exchange, while Figure 19 shows an example of such an exchange.

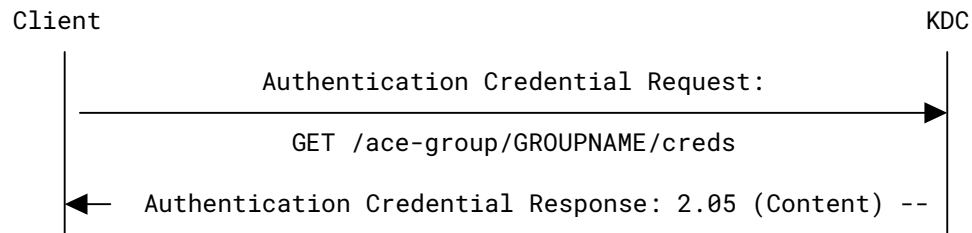


Figure 18: Message Flow of Authentication Credential Request-Response to Obtain the Authentication Credentials of All the Group Members

```

Request:

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

Response:

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / num /          9: 12,
  / creds /       13: [h' a2026008a101a5010202410220012158
                    20cd4177ba62433375ede279b5e18e8b
                    91bc3ed8f1e174474a26fc0edb44ea53
                    73225820a0391de29c5c5badda610d4e
                    301eaaa18422367722289cd18cbe6624
                    e89b9cfd',
                    h' a2026008a101a5010202410320012158
                    20ac75e9ece3e50bfc8ed60399889522
                    405c47bf16df96660a41298cb4307f7e
                    b62258206e5de611388a4b8a8211334a
                    c7d37ecb52a387d257e6db3c2a93df21
                    ff3affc8',
                    h' a2026008a101a5010202410920012158
                    206f9702a66602d78f5e81bac1e0af01
                    f8b52810c502e87ebb7c926c07426fd0
                    2f225820c8d33274c71c9b3ee57d842b
                    bf2238b8283cb410eca216fb72a78ea7
                    a870f800' ],
  / peer_roles /   14: ["sender", ["sender", "receiver"],
                    "receiver"],
  / peer_identifiers / 15: [h'01', h'02', h'03']
}

```

Figure 19: Example of Authentication Credential Request-Response to Obtain the Authentication Credentials of All the Group Members

4.5. /ace-group/GROUPNAME/kdc-cred

This resource implements a GET handler.

4.5.1. GET Handler

The handler expects a GET request.

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

- 'kdc_cred': specifying the KDC's authentication credential. This parameter is encoded like the 'kdc_cred' parameter in the Join Response (see [Section 4.3.1](#)).
- 'kdc_nonce': specifying a nonce generated by the KDC. This parameter is encoded like the 'kdc_nonce' parameter in the Join Response (see [Section 4.3.1](#)).
- 'kdc_cred_verify': specifying the PoP evidence computed by the KDC over the following PoP input: the nonce N_C (encoded as a CBOR byte string) concatenated with the nonce N_KDC (encoded as a CBOR byte string), where:
 - N_C is the nonce generated by the Client group member such that: i) the nonce was specified in the 'nonce' parameter of the latest Join Request that the Client sent to the KDC in order to join the group identified by GROUPNAME; and ii) the KDC stored the nonce as a 'clientchallenge' value associated with the Client after sending the corresponding Join Response (see [Section 4.3.1](#)).
 - N_KDC is the nonce generated by the KDC and specified in the 'kdc_nonce' parameter.

An example of PoP input to compute 'kdc_cred_verify' using CBOR encoding is given in [Figure 20](#).

The PoP evidence is computed by means of the same method used for computing the PoP evidence that was included in the Join Response for this Client (see [Section 4.3.1](#)).

Application profiles of this specification **MUST** specify the exact approaches used by the KDC to compute the PoP evidence to include in the 'kdc_cred_verify' parameter and **MUST** specify which of those approaches is used in which case ([REQ21](#)).

If an application profile supports the presence of external signature verifiers that send GET requests to this resource, then the application profile **MUST** specify how external signature verifiers provide the KDC with a self-generated nonce to use as N_C ([REQ21](#)).

```
N_C and N_KDC expressed in CBOR diagnostic notation:
```

```
N_C   = h'25a8991cd700ac01'  
N_KDC = h'0b7db12aaff56da3'
```

```
N_C and N_KDC as CBOR encoded byte strings:
```

```
N_C   = 0x4825a8991cd700ac01  
N_KDC = 0x480b7db12aaff56da3
```

```
PoP input:
```

```
0x48 25a8991cd700ac01 48 0b7db12aaff56da3
```

Figure 20: Example of PoP Input to Compute 'kdc_cred_verify' Using CBOR Encoding

4.5.1.1. Retrieve the KDC's Authentication Credential

In case the KDC has an associated authentication credential as required for the correct group operation, a group member or an external signature verifier can contact the KDC to request the KDC's authentication credential by sending a CoAP GET request to the `/ace-group/GROUPNAME/kdc-cred` endpoint at the KDC, where `GROUPNAME` identifies the group.

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

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

[Figure 21](#) gives an overview of the exchange described above, while [Figure 22](#) shows an example.

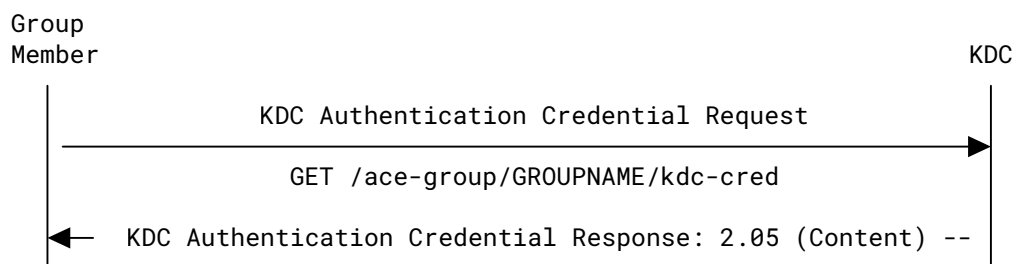


Figure 21: Message Flow of KDC Authentication Credential Request-Response to Obtain the Authentication Credential of the KDC

```

Request :

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

Response :

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / kdc_cred /      17: h'a2026008a101a5010202419920012158
                    2065eda5a12577c2bae829437fe33870
                    1a10aaa375e1bb5b5de108de439c0855
                    1d2258201e52ed75701163f7f9e40ddf
                    9f341b3dc9ba860af7e0ca7ca7e9eecd
                    0084d19c',
  / kdc_nonce /    18: h'0b7db12aaff56da3',
  / kdc_cred_verify / 19: h'3fc54702aa56e1b2cb20284294c9106a
                    63f91bac658d69351210a031d8fc7c5f
                    f3e4be39445b1a3e83e1510d1aca2f2e
                    8a7c081c7645042b18aba9d1fad1bd9c'
}

```

Figure 22: Example of KDC Authentication Credential Request-Response to Obtain the Authentication Credential of the KDC

4.6. /ace-group/GROUPNAME/policies

This resource implements the GET handler.

4.6.1. GET Handler

The handler expects a GET request.

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

If all verifications succeed, the handler replies with a 2.05 (Content) response containing the list of policies for the group identified by GROUPNAME. The payload of the response is formatted as a CBOR map including only the 'group_policies' parameter defined in [Section 4.3.1](#) and specifying the current policies in the group. If the KDC does not store any policy, the payload is formatted as a CBOR byte string of zero length (0x40).

The specific format and meaning of group policies **MUST** be specified in application profiles of this specification (REQ20).

4.6.1.1. Retrieve the Group Policies

A node in the group can contact the KDC to retrieve the current group policies by sending a CoAP GET request to the `/ace-group/GROUPNAME/policies` endpoint at the KDC, which is formatted as defined in Section 4.6.1 and where GROUPNAME identifies the group.

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

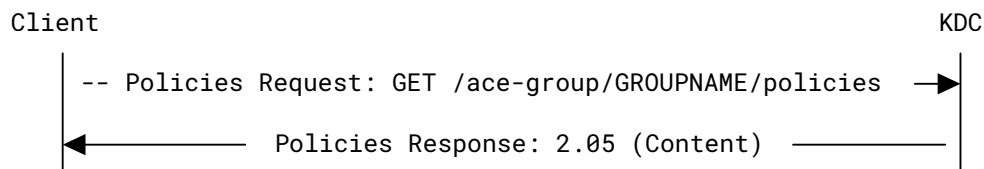


Figure 23: Message Flow of Policies Request-Response

```

Request:

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

Response:

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload(in CBOR diagnostic notation):
{
  / group_policies / 16: {
    / Expiration Delta / 2: 120
  }
}
  
```

Figure 24: Example of Policies Request-Response

4.7. `/ace-group/GROUPNAME/num`

This resource implements the GET handler.

4.7.1. GET Handler

The handler expects a GET request.

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

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

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

4.7.1.1. Retrieve the Keying Material Version

A node in the group can contact the KDC to request information about the version number of the symmetric group keying material by sending a CoAP GET request to the /ace-group/GROUPNAME/num endpoint at the KDC, which is formatted as defined in [Section 4.7.1](#) and where GROUPNAME identifies the group. In particular, the version is incremented by the KDC every time the group keying material is renewed before it is distributed to the group members.

[Figure 25](#) gives an overview of the exchange described above, while [Figure 26](#) shows an example.

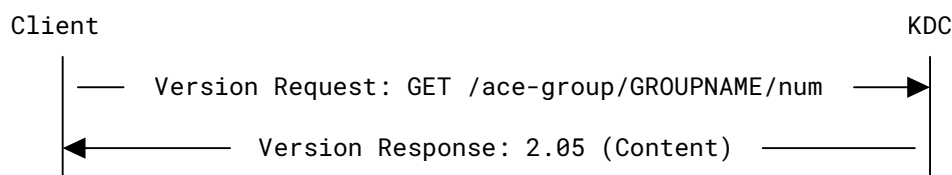


Figure 25: Message Flow of Version Request-Response

```
Request :

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

Response :

Header: Content (Code=2.05)
Content-Format: 60 (application/cbor)
Payload (in CBOR diagnostic notation):
  13
```

Figure 26: Example of Version Request-Response

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

This resource implements the GET, POST, and DELETE handlers.

In addition to what is defined in [Section 4.1.2](#), each of the handlers performs the following two verifications.

- The handler verifies that the Client is a current member of the group. If the verification fails, the KDC **MUST** reply with a 4.03 (Forbidden) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 0 ("Operation permitted only to group members").
- The handler verifies that the node name of the Client is equal to NODENAME used in the url-path. If the verification fails, the handler replies with a 4.03 (Forbidden) error response.

4.8.1. GET Handler

The handler expects a GET request.

If all verifications succeed, the handler replies with a 2.05 (Content) response containing both the group keying material and the individual keying material for the Client or information enabling the Client to derive it.

The payload of the response is formatted as a CBOR map, which includes the same fields of the response defined in [Section 4.3.2](#). In particular, the format for the group keying material is the same as defined in the response of [Section 4.3.2](#). If the 'exp' parameter is included, the 'exi' parameter **MUST** also be included. If the parameter 'exi' is included, its value specifies the residual lifetime of the group keying material from the current time at the KDC.

The CBOR map can include additional parameters that specify the individual keying material for the Client. The specific format of individual keying material for group members or of the information to derive such keying material **MUST** be defined in application profiles of this specification ([REQ27](#)), together with the corresponding CBOR map key that has to be registered in the "ACE Groupcomm Parameters" registry defined in [Section 11.7](#).

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

Note that the node could have also been observing the resource at `/ace-group/GROUPNAME` in order to be informed of changes in the group 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 CoAP No-Response Option [[RFC7967](#)] can use it when starting the observation of the sub-resource at `/ace-group/GROUPNAME/nodes/NODENAME`. In particular, the GET observation request can also include the No-Response option, with value set to 2 (Not interested in 2.xx responses).

4.8.1.1. Retrieve Group and Individual Keying Material

When any of the following happens, a node **MUST** stop using the stored 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 is indicated by the KDC through the 'exp' and/or 'exi' parameter (e.g., in a Join 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 Client has to request the latest keying material from the KDC. To this end, the Client sends a CoAP GET request to the `/ace-group/GROUPNAME/nodes/NODENAME` endpoint at the KDC, formatted as specified in [Section 4.8.1](#). The Client can request the latest keying material from the KDC before the currently stored, old keying material reaches its expiration time.

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

It is application dependent and pertaining to the used secure message exchange (e.g., [[GROUP-OSCORE](#)]) to set up these policies for instructing Clients to retain incoming messages and for how long ([OPT11](#)). This allows Clients to possibly decrypt such messages after getting updated keying material, rather than just consider them invalid messages to discard right away.

After having failed to decrypt messages from another group member and having sent a Key Distribution Request to the KDC, the Client might end up retrieving the same, latest group keying material that it already stores. In such a case, multiple failed decryptions might be due to the message sender and/or the KDC that have changed their authentication credential. Hence, the Client can retrieve such latest authentication credentials by sending to the KDC an Authentication Credential Request (see Sections 4.4.1.1 and 4.4.2.1) and a KDC Authentication Credential Request (see Section 4.5.1.1), respectively.

The Client can also send to the KDC a Key Distribution Request without having been triggered by a failed decryption of a message from another group member, if the Client wants to be sure that it currently stores the latest group keying material. If that is the case, the Client will receive from the KDC the same group keying material it already stores.

Figure 27 gives an overview of the exchange described above, while Figure 28 shows an example.

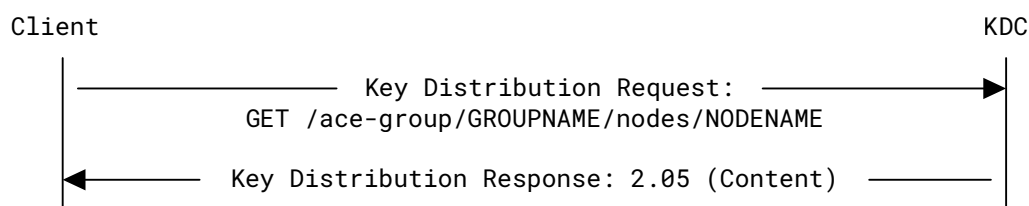


Figure 27: Message Flow of Key Distribution Request-Response

```
Request :

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

Response:

Header: Content (Code=2.05)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation, with "ind-key" being the
        profile-specified label for individual keying material):
{
  / gkty / 7: 65600,
  / key / 8: h'7365737369666e6b6579',
  / num / 9: 12,
  "ind-key": h'fcae9023'
}
```

Figure 28: Example of Key Distribution Request-Response

4.8.2. POST Handler

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

The handler expects a POST request with an empty payload.

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

If the KDC is currently not able to serve this request, i.e., to generate new individual keying material for the requesting Client, the KDC **MUST** reply with a 5.03 (Service unavailable) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 4 ("No available individual keying material").

If all verifications succeed, the handler replies with a 2.04 (Changed) response containing newly generated individual keying material for the Client. The payload of the response is formatted as a CBOR map. The specific format of newly generated individual keying material for group

members or of the information to derive such keying material **MUST** be defined in application profiles of this specification ([REQ27](#)), together with the corresponding CBOR map key that has to be registered in the "ACE Groupcomm Parameters" registry defined in [Section 11.7](#).

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

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

In either case, the handler may specify the new group keying material as part of the 2.04 (Changed) response.

Note that this handler is not intended to accommodate requests from a group member to trigger a group rekeying, whose scheduling and execution is an exclusive prerogative of the KDC (also see related security considerations in [Section 10.2](#)).

4.8.2.1. Request to Change Individual Keying Material

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

To this end, the Client performs a Key Renewal Request-Response exchange with the KDC, i.e., it sends a CoAP POST request to the `/ace-group/GROUPNAME/nodes/NODENAME` endpoint at the KDC, which is formatted as defined in [Section 4.8.1](#), where GROUPNAME identifies the group and NODENAME is the node name of the Client.

[Figure 29](#) gives an overview of the exchange described above, while [Figure 30](#) shows an example.

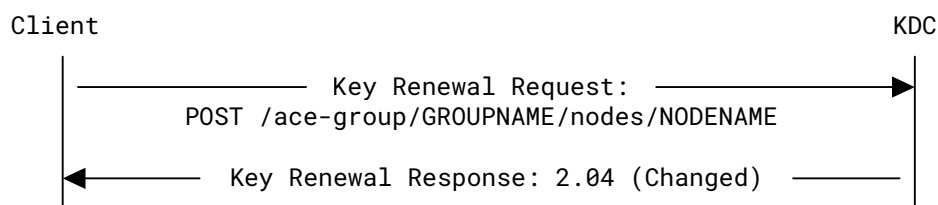


Figure 29: Message Flow of Key Renewal Request-Response

```
Request :

Header: POST (Code=0.02)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "nodes"
Uri-Path: "c101"

Response:

Header: Changed (Code=2.04)
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation, with "ind-key" being the
        profile-specified label for individual keying material):
{
  "ind-key": h'b71acc28'
}
```

Figure 30: Example of Key Renewal Request-Response

Note that there is a difference between the Key Renewal Request in this section and the Key Distribution Request in [Section 4.8.1.1](#). The former asks the KDC for new individual keying material, while the latter asks the KDC for the current group keying material together with the current individual keying material.

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

4.8.3. DELETE Handler

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

The handler expects a DELETE request with an empty payload.

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

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

4.8.3.1. Leave the Group

A Client can actively request to leave the group. In this case, the Client sends a CoAP DELETE request to the endpoint `/ace-group/GROUPNAME/nodes/NODENAME` at the KDC, where `GROUPNAME` identifies the group and `NODENAME` is the Client's node name.

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

4.9. `/ace-group/GROUPNAME/nodes/NODENAME/cred`

This resource implements the POST handler.

4.9.1. POST Handler

The POST handler is used to replace the stored authentication credential 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 the payload as specified in [Section 4.3.1](#), with the difference that the payload includes only the parameters `'client_cred'`, `'cnonce'`, and `'client_cred_verify'`.

The PoP evidence included in the `'client_cred_verify'` parameter is computed in the same way considered in [Section 4.3.1](#) and defined by the specific application profile ([REQ14](#)) by using the following to build the PoP input: i) the same scope entry specified by the Client in the `'scope'` parameter of the latest Join Request that the Client sent to the KDC in order to join the group identified by `GROUPNAME`; ii) the latest `N_S` value stored by the Client; and iii) a new `N_C` nonce generated by the Client and specified in the parameter `'cnonce'` of this request.

An example of PoP input to compute `'client_cred_verify'` using CBOR encoding is given in [Figure 31](#).

It is **REQUIRED** for application profiles to define the specific formats of authentication credentials that are acceptable to use in the group ([REQ6](#)).

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

If the KDC cannot retrieve the `'kdcchallenge'` associated with this Client (see [Section 3.3](#)), the KDC **MUST** reply with a 4.00 (Bad Request) error response, which **MUST** also have Content-Format `"application/ace-groupcomm+cbor"`. The payload of the error response is a CBOR map including

the 'kdcchallenge' parameter, which specifies a newly generated 'kdcchallenge' value. In such a case, the KDC **MUST** store the newly generated value as the 'kdcchallenge' value associated with this Client, replacing the currently stored value (if any).

Otherwise, the handler checks that the authentication credential specified in the 'client_cred' field is valid for the group identified by GROUPNAME. That is, the handler checks that the authentication credential is encoded according to the format used in the group, is intended for the public key algorithm used in the group, and is aligned with the possible associated parameters used in the group. If that cannot be successfully verified, the handler **MUST** reply with a 4.00 (Bad Request) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 2 ("Authentication credential incompatible with the group configuration").

Otherwise, the handler verifies the PoP evidence conveyed in the 'client_cred_verify' parameter of the request, by using the authentication credential specified in the 'client_cred' parameter as well as the same way considered in [Section 4.3.1](#) and defined by the specific application profile (REQ14). If the PoP evidence does not pass verification, the handler **MUST** reply with a 4.00 (Bad Request) error response. The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 3 ("Invalid proof-of-possession evidence").

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

- The handler associates the authentication credential from the 'client_cred' parameter of the request with the node identifier NODENAME, as well as with the access token associated with the node identified by NODENAME.
- In the stored list of group members' authentication credentials for the group identified by GROUPNAME, the handler replaces the authentication credential of the node identified by NODENAME with the authentication credential specified in the 'client_cred' parameter of the request.

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

```

scope, N_S, and N_C expressed in CBOR diagnostic notation:
  scope = h'826667726f7570316673656e646572'
  N_S   = h'018a278f7faab55a'
  N_C   = h'0446baefc56111bf'

scope, N_S, and N_C as CBOR encoded byte strings:
  scope = 0x4f826667726F7570316673656E646572
  N_S   = 0x48018a278f7faab55a
  N_C   = 0x480446baefc56111bf

PoP input:
  0x4f 826667726f7570316673656e646572
    48 018a278f7faab55a 48 0446baefc56111bf

```

Figure 31: Example of PoP Input to Compute 'client_cred_verify' Using CBOR Encoding

4.9.1.1. Uploading an Authentication Credential

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

To this end, the Client performs an Authentication Credential Update Request-Response exchange with the KDC, i.e., it sends a CoAP POST request to the `/ace-group/GROUPNAME/nodes/NODENAME/cred` endpoint at the KDC, where `GROUPNAME` identifies the group and `NODENAME` is the Client's node name.

The request is formatted as specified in [Section 4.9.1](#).

[Figure 32](#) gives an overview of the exchange described above, while [Figure 33](#) shows an example.

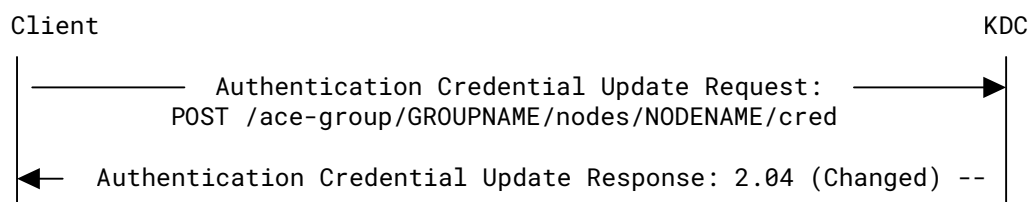


Figure 32: Message Flow of Authentication Credential Update Request-Response


```

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: "cred"
Content-Format: 261 (application/ace-groupcomm+cbor)
Payload (in CBOR diagnostic notation):
{
  / client_cred /          5: h' a2026008a101a501020241fc20012158
                           20bac5b11cad8f99f9c72b05cf4b9e26
                           d244dc189f745228255a219a86d6a09e
                           ff22582020138bf82dc1b6d562be0fa5
                           4ab7804a3a64b6d72ccfed6b6fb6ed28
                           bbfc117e',
  / cnonce /              6: h' 0446baefc56111bf',
  / client_cred_verify / 24: h' e2aeafd40d69d19dfe6e52077c5d7ff4
                              e408282cbefb5d06cbf414af2e19d982
                              ac45ac98b8544c908b4507de1e90b717
                              c3d34816fe926a2b98f53afd2fa0f30a'
}

Response:

Header: Changed (Code=2.04)

```

Figure 33: Example of Authentication Credential Update Request-Response

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

5. Removal of a Group Member

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

1. The Client explicitly asks to leave the group, as defined in [Section 4.8.3.1](#).
2. The node has been found compromised or is suspected so. The KDC is expected to determine that a group member has to be evicted either through its own means or based on information that it obtains from a trusted source (e.g., an Intrusion Detection System or an issuer of authentication credentials). Additional mechanics, protocols, and interfaces at the KDC that can support this are out of the scope of this document.
3. The Client's authorization to be a group member with the current roles is not valid anymore, i.e., the access token has expired or has been revoked. If the AS provides token introspection

(see [Section 5.9](#) of [\[RFC9200\]](#)), the KDC can optionally use it and check whether the Client is still authorized.

In all cases, the KDC performs the following actions.

- The KDC removes the Client from the list of current members of the group. When doing so, the KDC deletes the currently stored value of 'clientchallenge' for that Client, which was specified in the latest Join Request that the Client sent to the KDC in order to join the group (see [Section 4.3.1](#)).
- In case of forced eviction, i.e., for cases 2 and 3 above, the KDC deletes the authentication credential of the removed Client if it acts as a repository of authentication credentials for group members.
- If the removed Client is registered as an observer of the group-membership resource at /ace-group/GROUPNAME, the KDC removes the Client from the list of observers of that resource.
- If the sub-resource /nodes/NODENAME was created for the removed Client, the KDC deletes that sub-resource.

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

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

The response **MUST** have Content-Format "application/concise-problem-details+cbor" and is formatted as defined in [Section 4.1.2](#). Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 5 ("Group membership terminated").

- If forward security is prescribed by application policies installed at the KDC or by the used application profile of this specification, then the KDC **MUST** generate new group keying material and securely distribute it to all the current group members except the leaving node (see [Section 6](#)).

6. Group Rekeying Process

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

The KDC can perform a group rekeying before the current group keying material expires, unless it is acceptable or there are reasons to temporarily pause secure communications in the group, following the expiration of the current keying material. For example, a pause in the group communication might have been scheduled to start anyway when the group keying material expires, e.g., to allow maintenance operations on the group members. As another example, the KDC might be carrying out a verification that some group members are seemingly compromised and to be evicted, and this needs to be completed in order to appropriately define and schedule the exact rekeying process to perform. As a result, the KDC could delay the execution of the group rekeying.

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

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

Each rekeying message **MUST** have Content-Format "application/ace-groupcomm+cbor" and its payload is formatted as a CBOR map, which **MUST** include at least the information specified in the Key Distribution Response message (see [Section 4.3.2](#)), i.e., the parameters 'gkty', 'key', and 'num' defined in [Section 4.3.1](#). The CBOR map **SHOULD** also include the parameters 'exp' and 'exi'. If the 'exp' parameter is included, the 'exi' parameter **MUST** also be included. The CBOR map **MAY** include the parameter 'mgt_key_material' to specify new administrative keying material for the target group members if it is relevant for the used rekeying scheme.

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

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

Consistently with the used group rekeying scheme, the actual delivery of rekeying messages can occur through different approaches, as discussed in Sections 6.1 and 6.2.

The possible, temporary misalignment of the keying material stored by the different group members due to a group rekeying is discussed in Section 6.3. Further security considerations related to the group rekeying process are compiled in Section 10.2.

6.1. Point-to-Point Group Rekeying

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

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

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

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

In case the KDC deletes the group (and thus deletes the /ace-group/GROUPNAME resource), relying on CoAP Observe as discussed above 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 "application/concise-problem-details+cbor" and is formatted as defined in Section 4.1.2. Within the Custom Problem Detail entry 'ace-groupcomm-error', the value of the 'error-id' field **MUST** be set to 6 ("Group deleted").

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

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

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

This yields an overall lower number of rekeying messages, thus potentially reducing the overall time required to rekey the group. On the other hand, it requires the KDC to provide and use additional administrative keying material to protect the rekeying messages and to additionally sign them to ensure source authentication (see [Section 6.2.1](#)).

Compared to a group rekeying performed in a point-to-point fashion (see [Section 6.1](#)), a one-to-many group rekeying typically pays off in large-scale groups due to the reduced time for completing the rekeying, a more efficient utilization of network resources, and a reduced performance overhead at the KDC. To different extents, it also requires individual group members to locally perform additional operations in order to handle the administrative keying material and verify source authentication of rekeying messages. Therefore, one-to-many group rekeying schemes and their employment ought to ensure that the experienced performance overhead on the group members also remains bearable for resource-constrained devices.

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

When one of these group rekeying schemes is used, the KDC provides related information to a Client joining the group in the Join Response message (see [Section 4.3.1](#)). In particular, the 'rekeying_scheme' parameter indicates the rekeying scheme used in the group (if no default scheme can be assumed); the 'control_group_uri' parameter, if present, specifies a URI whose addressing information is, e.g., a multicast IP address where the KDC will send the rekeying messages for that group as intended to reach all the group members; and the 'mgt_key_material' parameter specifies a subset of the administrative keying material intended for that particular joining Client to have, as used to protect the rekeying messages sent to the group when also intended for that joining Client.

Rekeying messages can be protected at the application layer by using COSE [[RFC9052](#)] and the administrative keying material as prescribed by the specific group rekeying scheme (see [Section 6.2.1](#)). After that, the delivery of protected rekeying messages to the intended target group members can occur in different ways, such as the following ones.

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

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

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

In order to make such message delivery more efficient, the rekeying topic associated with a group can be further organized into subtopics. For instance, the KDC can use a particular subtopic to address a particular set of target group members during the rekeying process as possibly aligned to a similar organization of the administrative keying material (e.g., a key hierarchy).

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

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

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

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

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

[Figure 35](#) provides an example of a one-to-many group rekeying over multicast. In particular, the example makes the following assumptions:

- The group currently consists of four group members, namely C1, C2, C3, and C4.
- Each group member, when joining the group, provided the KDC with a URI in the 'control_uri' parameter with url-path "grp-rek".

- Each group member, when joining the group, received from the KDC a URI in the 'control_group_uri' parameter, specifying the multicast address MULT_ADDR and url-path "grp-mrek".
- Before the group rekeying is performed, the keying material used in the group has version number num=5.
- The KDC performs the group rekeying in such a way to evict the group member C3, which has been found to be compromised.

In the example, the KDC determines that the most convenient way to perform a group rekeying that evicts C3 is as follows.

First, the KDC sends one rekeying message over multicast to the multicast address MULT_ADDR and the url-path "grp-mrek". In the figure, the message is denoted with solid arrows. The message is protected with a non-compromised key from the administrative keying material that only C1 and C2 store. Therefore, even though all the group members receive this message, only C1 and C2 are able to decrypt it. The message includes: the new group keying material with version number num=6 and new keys from the administrative keying material to replace those stored by the group members C1, C2, and C3.

After that, the KDC sends one rekeying message addressed individually to C4 and with url-path "grp-rek". In the figure, the message is denoted with a dotted arrow. The message is protected with the secure association shared between C4 and the KDC. The message includes: the new group keying material with version number num=6 and new keys from the administrative keying material to replace those stored by both C4 and C3.

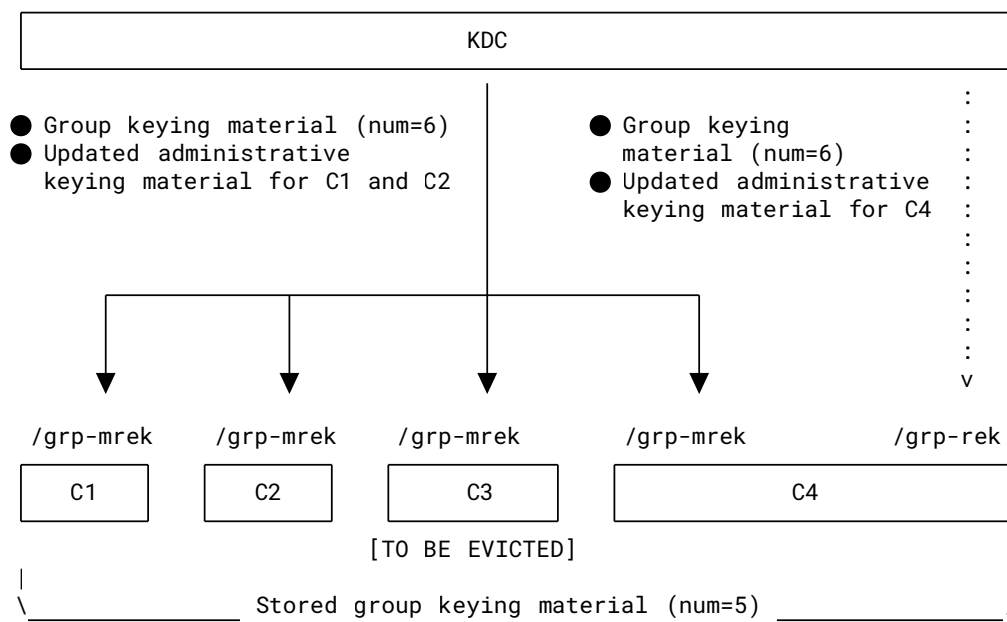


Figure 35: Example of Message Exchanges for a One-to-Many Group Rekeying

6.2.1. Protection of Rekeying Messages

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

The following describes one possible method for the KDC to protect the rekeying messages when using the administrative keying material.

The method assumes that the following holds for the administrative keying material specified in the 'mgt_key_material' parameter of the Join Response (see [Section 4.3.1](#)).

- The encryption algorithm **SHOULD** be the same one used to protect communications in the group.
- The included symmetric encryption keys are accompanied by a corresponding and unique key identifier assigned by the KDC.
- A Base IV is also included with the same size of the AEAD nonce considered by the encryption algorithm to use.

First, the KDC computes a COSE_Encrypt0 object as follows.

- The encryption key to use is selected from the administrative keying material, as defined by the rekeying scheme used in the group.
- The plaintext is the actual data content of the current rekeying message.
- The Additional Authenticated Data (AAD) is empty unless otherwise specified by separate documents profiling the use of the group rekeying scheme.

- Since the KDC is the only sender of rekeying messages, the AEAD nonce can be computed as follows, where `NONCE_SIZE` is the size in bytes of the AEAD nonce. Separate documents profiling the use of the group rekeying scheme may define alternative ways to compute the AEAD nonce.

The KDC considers the following values.

- `COUNT`: as a 2-byte unsigned integer associated with the used encryption key. Its value is set to 0 when starting to perform a new group rekeying instance and is incremented after each use of the encryption key.
- `NEW_NUM`: as the version number of the new group keying material to distribute in this rekeying instance, left-padded with zeros to exactly `NONCE_SIZE - 2` bytes.

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

In order to comply with the security requirements of AEAD encryption algorithms, the KDC **MUST NOT** reuse the same pair (AEAD encryption key, AEAD nonce). For example, this includes not using the same encryption key from the administrative keying material more than 2^{16} times during the same rekeying instance.

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

In order to ensure source authentication, each rekeying message protected with the administrative keying material **MUST** be signed by the KDC. To this end, the KDC computes a countersignature of the `COSE_Encrypt0` object, as described in Sections 3.2 and 3.3 of [RFC9338]. In particular, the following applies when computing the countersignature.

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

If the source authentication of messages exchanged in the group is also ensured by means of signatures, then rekeying messages **MUST** be signed using the same signature algorithm and related parameters. Also, the KDC's authentication credential including the public key to use for

signature verification **MUST** be provided in the Join Response through the 'kdc_cred' parameter, together with the corresponding proof-of-possession (PoP) evidence in the 'kdc_cred_verify' parameter.

If source authentication of messages exchanged in the group is not ensured by means of signatures, then the administrative keying material conveyed in the 'mgt_key_material' parameter of the Join Response sent by KDC (see [Section 4.3.1](#)) **MUST** also comprise a KDC's authentication credential including the public key to use for signature verification, together with the corresponding PoP evidence. Within the 'mgt_key_material' parameter, it is **RECOMMENDED** to specify this information by using the same format and encoding used for the parameters 'kdc_cred', 'kdc_nonce', and 'kdc_cred_verify' in the Join Response. It is up to separate documents profiling the use of the group rekeying scheme to specify such details.

After that, the KDC specifies the computed countersignature in the 'Countersignature0 version 2' header parameter of the COSE_Encrypt0 object.

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

6.3. Misalignment of Group Keying Material

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

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

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

7. Extended Scope Format

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

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

The extended format is intended only for the 'scope' claim of access tokens for the cases where the claim takes a CBOR byte string as the value. 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 [\[RFC9200\]](#)), the AS Request Creation Hints message from the Resource Server (see [Section 5.3](#) of [\[RFC9200\]](#)), and the Introspection Response from the Authorization Server (see [Section 5.9.2](#) of [\[RFC9200\]](#)).

The value of the 'scope' claim following the extended format is composed as follows. Given the original scope using semantics SEM and encoded as a CBOR byte string, the corresponding extended scope consists of the same CBOR byte string enclosed by a CBOR tag [\[RFC8949\]](#), whose tag number identifies the semantics SEM.

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

Figures [36](#) and [37](#) build on the examples in [Section 3.1](#) and show the corresponding extended scopes.

```

;# include rfc9237

gname = tstr

permissions = uint .bits roles

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

scope_entries = AIF-Generic<gname, permissions>

scope = bstr .cbor scope_entries

extended_scope = #6.<TAG_FOR_THIS_SEMANTICS>(scope)

TAG_FOR_THIS_SEMANTICS = uint

```

Figure 36: Example of Extended scope Using AIF

```

gname = tstr

role = tstr

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

scope_entries = [ * scope_entry ]

scope = bstr .cbor scope_entries

extended_scope = #6.<TAG_FOR_THIS_SEMANTICS>(scope)

TAG_FOR_THIS_SEMANTICS = uint

```

Figure 37: Example of Extended scope Using the Textual Format, with the Role Identifiers Encoded as Text Strings

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.

Applications and application profiles using the extended format of scope have to specify which CBOR tag from [CBOR.Tags] is used for identifying the scope semantics or to register a new CBOR tag if a suitable one does not exist already (REQ28). In case there is an already existing, suitable CBOR tag, a new CBOR tag should not be registered in order to avoid code point squatting.

If the binary-encoded scope uses semantics associated with a registered CoAP Content-Format [RFC7252] [CoAP.Content.Formats], then a suitable CBOR tag associated with that CoAP Content-Format would already be registered, as defined in Section 4.3 of [RFC9277].

This is especially relevant when the binary encoded scope uses AIF. That is, it is expected that the definition of an AIF-specific data model comes together with the registration of CoAP Content-Formats for the relevant combinations of its Toid and Tperm values. As discussed above, this yields the automatic registration of the CBOR tags associated with those CoAP Content-Formats.

8. ACE Groupcomm Parameters

This specification defines a number of parameters used during the second phase of the key provisioning process, i.e., after the exchange after the exchange of Token Transfer Request and Response. The table below summarizes them and specifies the CBOR map keys to use instead of the full descriptive names.

Note that the media type "application/ace-groupcomm+cbor" **MUST** be used when these parameters are transported in the respective CBOR map entries.

Name	CBOR Key	CBOR Type	Reference
gid	0	array	RFC 9594
gname	1	array of tstr	RFC 9594
guri	2	array of tstr	RFC 9594
scope	3	bstr	RFC 9594
get_creds	4	Null or array	RFC 9594
client_cred	5	bstr	RFC 9594
cnonce	6	bstr	RFC 9594
gkty	7	int or tstr	RFC 9594
key	8	See the "ACE Groupcomm Key Types" registry	RFC 9594
num	9	int	RFC 9594
ace_groupcomm_profile	10	int	RFC 9594
exp	11	uint	RFC 9594
exi	12	uint	RFC 9594

Name	CBOR Key	CBOR Type	Reference
creds	13	array	RFC 9594
peer_roles	14	array	RFC 9594
peer_identifiers	15	array	RFC 9594
group_policies	16	map	RFC 9594
kdc_cred	17	bstr	RFC 9594
kdc_nonce	18	bstr	RFC 9594
kdc_cred_verify	19	bstr	RFC 9594
rekeying_scheme	20	int	RFC 9594
client_cred_verify	24	bstr	RFC 9594
creds_repo	25	tstr	RFC 9594
control_uri	26	tstr	RFC 9594
mgt_key_material	27	bstr	RFC 9594
control_group_uri	28	tstr	RFC 9594
sign_info	29	Null or array	RFC 9594
kdcchallenge	30	bstr	RFC 9594

Table 5: ACE Groupcomm Parameters

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

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

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

A Client **MUST** support the following parameters.

- 'scope'
- 'cnonce'
- 'gkty'
- 'key'
- 'num'
- 'exp'
- 'exi'
- 'gid'
- 'gname'
- 'guri'
- 'creds'
- 'peer_identifiers'
- 'ace_groupcomm_profile'
- 'control_uri'
- 'rekeying_scheme'

A Client **SHOULD** support the following parameter.

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

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

- 'client_cred' and 'client_cred_verify': These parameters are relevant for a Client that has an authentication credential to use in a joined group.
- 'kdcchallenge': This parameter is relevant for a Client that has an authentication credential to use in a joined group and that provides the access token to the KDC through a Token Transfer Request (see [Section 3.3](#)).
- 'creds_repo': This parameter is relevant for a Client that has an authentication credential to use in a joined group and that makes it available from a key repository different than the KDC.
- 'group_policies': This parameter is relevant for a Client that is interested in the specific policies used in a group, but it does not know them or cannot become aware of them before joining that group.

- 'peer_roles': This parameter is relevant for a Client that has to know about the roles of other group members, especially when retrieving and handling their corresponding authentication credentials.
- 'kdc_nonce', 'kdc_cred', and 'kdc_cred_verify': These parameters are relevant for a Client that joins a group for which, as per the used application profile of this specification, the KDC has an associated authentication credential and this is required for the correct group operation.
- 'mgt_key_material': This parameter is relevant for a Client that supports an advanced rekeying scheme possibly used in the group, such as based on one-to-many rekeying messages sent over IP multicast.
- 'control_group_uri': This parameter is relevant for a Client that also acts as a CoAP server supporting: i) the hosting of a dedicated resource for each group that the Client is interested to be a part of; and ii) the reception of one-to-many requests sent to those resources by the KDC (e.g., over IP multicast), as targeting multiple members of the corresponding group. Examples of related management operations that the KDC can perform by this means are the eviction of group members and the execution of a group rekeying process through an advanced rekeying scheme, such as based on one-to-many rekeying messages.

9. ACE Groupcomm Error Identifiers

This specification defines a number of values that the KDC can use as error identifiers. These are used in error responses with Content-Format "application/concise-problem-details+cbor", as values of the 'error-id' field within the Custom Problem Detail entry 'ace-groupcomm-error' (see [Section 4.1.2](#)).

Value	Description
0	Operation permitted only to group members
1	Request inconsistent with the current roles
2	Authentication credential incompatible with the group configuration
3	Invalid proof-of-possession evidence
4	No available individual keying material
5	Group membership terminated
6	Group deleted

Table 6: ACE Groupcomm Error Identifiers

If a Client supports the problem-details format [[RFC9290](#)] and the Custom Problem Detail entry 'ace-groupcomm-error' defined in [Section 4.1.2](#) of this document and is able to understand the error specified in the 'error-id' field therein, then the Client can use that information to

determine what actions to take next. If the Concise Problem Details data item specified in the error response includes the 'detail' entry and the Client supports it, such an entry may provide additional context.

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

- In case of error 0, the Client should stop sending the request in question to the KDC. Rather, the Client should first join the targeted group. If it has not happened already, this first requires the Client to obtain an appropriate access token authorizing access to the group and provide it to the KDC.
- In case of error 1, the Client as a group member should rejoin the group with all the roles needed to perform the operation in question. This might require the Client to first obtain a new access token and provide it to the KDC, if the current access token does not authorize the Client to take those roles in the group. For operations admitted to a Client that is not a group member (e.g., an external signature verifier), the Client should first obtain a new access token authorizing to also have the missing roles.
- In case of error 2, the Client has to obtain or self-generate a different asymmetric key pair, as aligned to the public key algorithm and parameters used in the targeted group. After that, the Client should provide the KDC with its new authentication credential, which is consistent with the format used in the targeted group and including the new public key.
- In case of error 3, the Client should ensure to compute its proof-of-possession evidence by correctly using the parameters and procedures defined in the used application profile of this specification. In an unattended setup, it might not be possible for a Client to autonomously diagnose the error and take an effective next action to address it.
- In case of error 4, the Client should wait for a certain (pre-configured) amount of time before trying to resend its request to the KDC.
- In case of error 5, the Client may try joining the group again. This might require the Client to first obtain a new access token and provide it to the KDC, e.g., if the current access token has expired.
- In case of error 6, the Client should clean up its state regarding the group, just like if it has left the group with no intention to rejoin it.

10. Security Considerations

Security considerations are inherited from the ACE framework [RFC9200] and from the specific transport profile of ACE used between the Clients and the KDC, e.g., [RFC9202] and [RFC9203].

When using the problem-details format defined in [RFC9290] for error responses, then the privacy and security considerations from Sections 4 and 5 of [RFC9290] also apply.

Furthermore, the following security considerations apply.

10.1. Secure Communication in the Group

When a group member receives a message from a certain sender for the first time since joining the group, it needs to have a mechanism in place to avoid replayed messages and to assert their freshness, e.g., as described in [Appendix B.1.2](#) of [RFC8613] or [Section 10](#) of [GROUP-OSCORE]. Such a mechanism also aids the recipient group member in case it has rebooted and lost the security state used to protect previous group communications with that sender.

By its nature, the KDC is invested with a large amount of trust, since it acts as a generator and provider of the symmetric keying material used to protect communications in each of its groups. While details depend on the specific communication and security protocols used in the group, the KDC is in the position to decrypt messages exchanged in the group as if it was also a group member, as long as those are protected through commonly shared group keying material.

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

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

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

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

10.2. Update of Group Keying Material

The KDC can generate new group keying material and provide it to the group members (rekeying) through the rekeying scheme used in the group, as discussed in [Section 6](#).

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

Unless otherwise defined by an application profile of this specification, the KDC **SHOULD** renew the group keying material upon a group membership change. As a possible exception, the KDC may not rekey the group upon the joining of a new group member if the application does not require backward security. As another possible exception discussed more in detail later in this section, the KDC may rely on a rekeying policy that reasonably takes into account the expected rate of group membership changes and the duration of a group rekeying.

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

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

Once a joining node has obtained the new, latest keying material through a Join Response from the KDC (see [Section 4.3.1.1](#)), the joining node becomes able to read any message that was exchanged in the group and protected with that keying material. This is the case if the KDC provides the current group members with the new, latest keying material before completing the joining procedure. However, the joining node is not able to read messages exchanged in the group and protected with keying material older than the one provided in the Join Response, i.e., having a strictly lower version number NUM.

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

The KDC may enforce a rekeying policy that takes into account the overall time required to rekey the group, as well as the expected rate of changes in the group membership. That is, the KDC may not rekey the group at each and every group membership change, for instance, if members' joining and leaving occur frequently and performing a group rekeying takes too long. Instead, the KDC might rekey the group after a minimum number of group members have joined or left within a given time interval, after a maximum amount of time since the last group rekeying was completed, or yet during predictable network inactivity periods.

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

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

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

However, if the KDC relies on Observe notifications to distribute the new group keying material, the KDC would have lost all the current ongoing Observations with the group members after rebooting, and the group members would continue using the old group keying material. Therefore, the KDC will rely on each group member asking for the new group keying material (see Sections 4.3.2.1 and 4.8.1.1) or perform a group rekeying by actively sending rekeying messages to group members as discussed in Section 6.

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

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

10.3. Block-Wise Considerations

If the Block-Wise CoAP options [RFC7959] are used and the keying material is updated in the middle of a Block-Wise transfer, the sender of the blocks just changes the group keying material to the updated one and continues the transfer. As long as both sides get the new group keying material, updating the group 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, and depending on how fast the group keying material is changed, the group members might consume a larger amount of the network bandwidth by repeatedly resending the same blocks, which might be problematic.

11. IANA Considerations

Per this document, IANA has completed the following actions.

11.1. Media Type Registrations

This specification has registered 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 a CBOR map containing the parameters defined in RFC 9594.

Security considerations: See [Section 10](#) of RFC 9594.

Interoperability considerations: N/A

Published specification: RFC 9594

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 RFC 9594.

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information: ACE WG mailing list (ace@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)

Intended usage: COMMON

Restrictions on usage: None

Author/Change controller: IETF

Provisional registration: No

11.2. CoAP Content-Formats

IANA has registered the following entry in the "CoAP Content-Formats" registry within the "CoRE Parameters" registry group.

Content Type: application/ace-groupcomm+cbor

Content Coding: -

ID: 261

Reference: RFC 9594

11.3. OAuth Parameters

IANA has registered the following entries in the "OAuth Parameters" registry, following the procedure specified in [Section 11.2](#) of [[RFC6749](#)].

Name: sign_info

Parameter Usage Location: client-rs request, rs-client response

Change Controller: IETF

Reference: RFC 9594

Name: kdcchallenge

Parameter Usage Location: rs-client response

Change Controller: IETF

Reference: RFC 9594

11.4. OAuth Parameters CBOR Mappings

IANA has registered the following entries in the "OAuth Parameters CBOR Mappings" registry, following the procedure specified in [Section 8.10](#) of [RFC9200].

Name: sign_info

CBOR Key: 45

Value Type: Null or array

Reference: RFC 9594

Name: kdcchallenge

CBOR Key: 46

Value Type: byte string

Reference: RFC 9594

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

IANA has registered the following entry in the "Interface Description (if=) Link Target Attribute Values" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Value: ace.groups

Description: The KDC interface at the parent resource of group-membership resources is used to retrieve names of security groups using the ACE framework.

Reference: [Section 4.1](#) of RFC 9594

Value: ace.group

Description: The KDC interface at a group-membership resource is used to provision keying material and related information and policies to members of the corresponding security group using the ACE framework.

Reference: [Section 4.1](#) of RFC 9594

11.6. Custom Problem Detail Keys Registry

IANA has registered the following entry in the "Custom Problem Detail Keys" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Key Value: 0

Name: ace-groupcomm-error

Brief Description: Carry RFC 9594 problem details in a Concise Problem Details data item.

Change Controller: IETF

Reference: RFC 9594, [Section 4.1.2](#)

11.7. ACE Groupcomm Parameters

This specification has established the "ACE Groupcomm Parameters" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#)

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 the CBOR map key of the item. These values **MUST** be unique. The value can be a positive integer, a negative integer, or a text string. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 as well as text strings of length 1 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 as well as text strings of length 2 are designated as "Specification Required". Integer values greater than 65535 as well as text strings of length greater than 2 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".

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

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

This registry has been initially populated with the values in [Table 5](#).

11.8. ACE Groupcomm Key Types

This specification establishes the "ACE Groupcomm Key Types" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

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. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 as well as text strings of length 1 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 as well as text strings of length 2 are designated as "Specification Required". Integer values greater than 65535 as well as text strings of length greater than 2 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".

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

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

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

This registry has been initially populated with the value in [Table 1](#).

11.9. ACE Groupcomm Profiles

This specification establishes the "ACE Groupcomm Profiles" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

The columns of this registry are:

Name: The name of the application profile.

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. These values **MUST** be unique. The value can be a positive integer or a negative integer. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". 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 field contains a pointer to the public specification for this application profile, if one exists.

This registry has been initially populated with the value in [Table 2](#).

11.10. ACE Groupcomm Policies

This specification establishes the "ACE Groupcomm Policies" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

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. These values **MUST** be unique. The value can be a positive integer, a negative integer, or a text string. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 as well as text strings of length 1 are designated as "Standards Action With Expert Review". Integer values from -65536 to -257 and from 256 to 65535 as well as text strings of length 2 are designated as "Specification Required". Integer values greater than 65535 as well as text 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 for this group communication policy and its format, if one exists.

This registry has been initially populated with the values in [Table 3](#).

11.11. Sequence Number Synchronization Methods

This specification establishes the "Sequence Number Synchronization Methods" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

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. These values **MUST** be unique. The value can be a positive integer, a negative integer, or a text string. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 as well as text strings of length 1 are designated as "Standards Action With Expert

Review". Integer values from -65536 to -257 and from 256 to 65535 as well as text strings of length 2 are designated as "Specification Required". Integer values greater than 65535 as well as text strings of length greater than 2 are designated as "Expert Review". Integer values less than -65536 are marked as "Private Use".

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

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

11.12. ACE Groupcomm Errors

This specification establishes the "ACE Groupcomm Errors" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

The columns of this registry are:

Value: The value to be used to identify the error. These values **MUST** be unique. The value can be a positive integer or a negative integer. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". 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".

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 with the values in [Table 6](#). The "Reference" column for all of these entries refers to this document.

11.13. ACE Groupcomm Rekeying Schemes

This specification establishes the "ACE Groupcomm Rekeying Schemes" IANA registry within the "Authentication and Authorization for Constrained Environments (ACE)" registry group.

Values in this registry are covered by different registration policies as indicated below. Some policies require Expert Review; guidelines are provided in [Section 11.14](#).

The columns of this registry are:

Value:

The value to be used to identify the group rekeying scheme. These values **MUST** be unique. The value can be a positive integer or a negative integer. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as "Standards Action With Expert Review". 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".

Name: The name of the group rekeying scheme.

Description: This field contains a brief description of the group rekeying scheme.

Reference: This field contains a pointer to the public specification defining the group rekeying scheme, if one exists.

This registry has been initially populated with the value in [Table 4](#).

11.14. 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. 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 assignments. 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 the device it will be used on, and the number of code points left that encode to that size.

12. References

12.1. Normative References

[CBOR.Tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <<https://www.iana.org/assignments/cbor-tags/>>.

-
- [CoAP.Content.Formats]** IANA, "CoAP Content-Formats", <<https://www.iana.org/assignments/core-parameters/>>.
- [COSE.Algorithms]** IANA, "COSE Algorithms", <<https://www.iana.org/assignments/cose/>>.
- [COSE.Header.Parameters]** IANA, "COSE Header Parameters", <<https://www.iana.org/assignments/cose/>>.
- [COSE.Key.Types]** IANA, "COSE Key Types", <<https://www.iana.org/assignments/cose/>>.
- [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>>.
- [RFC3629]** Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC6690]** Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749]** Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [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>>.

- [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>>.
- [RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <<https://www.rfc-editor.org/info/rfc9052>>.
- [RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <<https://www.rfc-editor.org/info/rfc9053>>.
- [RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <<https://www.rfc-editor.org/info/rfc9200>>.
- [RFC9237] Bormann, C., "An Authorization Information Format (AIF) for Authentication and Authorization for Constrained Environments (ACE)", RFC 9237, DOI 10.17487/RFC9237, August 2022, <<https://www.rfc-editor.org/info/rfc9237>>.
- [RFC9290] Fossati, T. and C. Bormann, "Concise Problem Details for Constrained Application Protocol (CoAP) APIs", RFC 9290, DOI 10.17487/RFC9290, October 2022, <<https://www.rfc-editor.org/info/rfc9290>>.
- [RFC9338] Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <<https://www.rfc-editor.org/info/rfc9338>>.

12.2. Informative References

- [C509-CERT] Preuß Mattsson, J., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-11, 8 July 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-11>>.
- [CoAP-PUBSUB] Jimenez, J., Koster, M., and A. Keränen, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-14, 18 April 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-14>>.
- [GROUP-CoAP] 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-11, 24 April 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-11>>.

-
- [GROUP-OSCORE]** Tiloca, M., Selander, G., Palombini, F., Preuß Mattsson, J., and R. Höglund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-22, 28 August 2024, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-22>>.
- [OSCORE-DISCOVERY]** Tiloca, M., Amsüss, C., and P. Van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-16, 4 September 2024, <<https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-discovery-16>>.
- [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>>.
- [RFC3986]** Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5280]** Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [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>>.
- [RFC9202] 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)", RFC 9202, DOI 10.17487/RFC9202, August 2022, <<https://www.rfc-editor.org/info/rfc9202>>.
- [RFC9203] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "The Object Security for Constrained RESTful Environments (OSCORE) Profile of the Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9203, DOI 10.17487/RFC9203, August 2022, <<https://www.rfc-editor.org/info/rfc9203>>.
- [RFC9277] Richardson, M. and C. Bormann, "On Stable Storage for Items in Concise Binary Object Representation (CBOR)", RFC 9277, DOI 10.17487/RFC9277, August 2022, <<https://www.rfc-editor.org/info/rfc9277>>.
- [RFC9431] Sengul, C. and A. Kirby, "Message Queuing Telemetry Transport (MQTT) and Transport Layer Security (TLS) Profile of Authentication and Authorization for Constrained Environments (ACE) Framework", RFC 9431, DOI 10.17487/RFC9431, July 2023, <<https://www.rfc-editor.org/info/rfc9431>>.

Appendix A. Requirements for Application Profiles

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

A.1. Mandatory-to-Address Requirements

- REQ1: Specify the format and encoding of scope. This includes defining the set of possible roles and their identifiers, as well as the corresponding encoding to use in the scope entries according to the used scope format (see [Section 3.1](#)).
- REQ2: If scope uses AIF, register its specific instance of "Toid" and "Tperm" as media type parameters and a corresponding Content-Format, as per the guidelines in [\[RFC9237\]](#).
- REQ3: If used, specify the acceptable values for the 'sign_alg' parameter (see [Section 3.3.1](#)).
- REQ4: If used, specify the acceptable values and structure for the 'sign_parameters' parameter (see [Section 3.3.1](#)).
- REQ5: If used, specify the acceptable values and structure for the 'sign_key_parameters' parameter (see [Section 3.3.1](#)).
- REQ6: Specify the acceptable formats for authentication credentials and, if applicable, the acceptable values for the 'cred_fmt' parameter (see [Section 3.3.1](#)).
- REQ7: If the value of the GROUPNAME URI path and the group name in the access token scope ('gname' in [Section 3.1](#)) are not required to coincide, specify the mechanism to map the GROUPNAME value in the URI to the group name (see [Section 4.1](#)).

-
- REQ8: Define whether the KDC has an authentication credential as required for the correct group operation and if this has to be provided through the 'kdc_cred' parameter (see Sections 4.1 and 4.3.1).
- REQ9: Specify if any part of the KDC interface as defined in this document is not supported by the KDC (see Section 4.1).
- REQ10: Register a Resource Type for the group-membership resources, which is used to discover the correct URL for sending a Join Request to the KDC (see Section 4.1).
- REQ11: Define what specific actions (e.g., CoAP methods) are allowed on each resource that are accessible through the KDC interface, depending on: whether the Client is a current group member; the roles that a Client is authorized to take as per the obtained access token (see Section 3.1); and the roles that the Client has as a current group member.
- REQ12: Categorize possible newly defined operations for Clients into primary operations expected to be minimally supported and secondary operations, and provide accompanying considerations (see Section 4.1.1).
- REQ13: Specify the encoding of group identifiers (see Section 4.2.1).
- REQ14: Specify the approaches used to compute and verify the PoP evidence to include in the 'client_cred_verify' parameter and which of those approaches is used in which case (see Section 4.3.1).
- REQ15: Specify how the nonce N_S is generated, if the access token is not provided to the KDC through the Token Transfer Request sent to the /authz-info endpoint (e.g., the access token is instead transferred during the establishment of a secure communication association).
- REQ16: Define the initial value of the version number for the group keying material (see Section 4.3.1).
- REQ17: Specify the format of the group keying material that is conveyed in the 'key' parameter (see Section 4.3.1).
- REQ18: Specify the acceptable values of the 'gkty' parameter (see Section 4.3.1). For each of them, register a corresponding entry in the "ACE Groupcomm Key Types" IANA registry if such an entry does not exist already.
- REQ19: Specify and register the application profile identifier (see Section 4.3.1).
- REQ20: If used, specify the format and default values of the entries of the CBOR map to include in the 'group_policies' parameter (see Section 4.3.1).
- REQ21: Specify the approaches used to compute and verify the PoP evidence to include in the 'kdc_cred_verify' parameter and which of those approaches is used in which case (see Sections 4.3.1 and 4.5.1). If external signature verifiers are supported, specify how those provide a nonce to the KDC to be used for computing the PoP evidence (see Section 4.5.1).
- REQ22: Specify the communication protocol that members of the group use to communicate with each other (e.g., CoAP for group communication).
- REQ23: Specify the security protocol that members of the group use to protect the group communication (e.g., Group OSCORE). This must provide encryption, integrity, and replay protection.
- REQ24: Specify how the communication is secured between the Client and the KDC. Optionally, specify a transport profile of ACE [RFC9200] to use between the Client and the KDC (see Section 4.3.1.1).
-

-
- REQ25: Specify the format of the node identifiers of group members (see Sections 4.3.1 and 4.4.1).
- REQ26: Specify policies at the KDC to handle node identifiers that are included in the 'get_creds' parameter but are not associated with any current group member (see Section 4.4.1).
- REQ27: Specify the format of (newly generated) individual keying material for group members or of the information to derive such keying material, as well as the corresponding CBOR map key that has to be registered in the "ACE Groupcomm Parameters" registry (see Sections 4.8.1 and 4.8.2).
- REQ28: Specify which CBOR tag is used for identifying the semantics of binary scopes, or register a new CBOR tag if a suitable one does not exist already (see Section 7).
- REQ29: Categorize newly defined parameters according to the same criteria of Section 8.
- REQ30: Define whether Clients must, should, or may support the conditional parameters defined in Section 8 and under which circumstances.

A.2. Optional-to-Address Requirements

- OPT1: Optionally, if the textual format of scope is used, specify CBOR values to use for abbreviating the role identifiers in the group (see Section 3.1).
- OPT2: Optionally, specify the additional parameters used in the exchange of Token Transfer Request and Response (see Section 3.3).
- OPT3: Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if the 'sign_info' parameter is not used (see Section 3.3).
- OPT4: Optionally, specify possible or required payload formats for specific error cases (see Section 4.1.2).
- OPT5: Optionally, specify additional identifiers of error types as values of the 'error-id' field within the Custom Problem Detail entry 'ace-groupcomm-error' (see Section 4.1.2).
- OPT6: Optionally, specify the encoding of the 'creds_repo' parameter if the default one is not used (see Section 4.3.1).
- OPT7: Optionally, specify the functionalities implemented at the resource hosted by the Client at the URI indicated in the 'control_uri' parameter, including the encoding of exchanged messages and other details (see Section 4.3.1).
- OPT8: Optionally, specify the behavior of the POST handler of group-membership resources, for the case when it fails to retrieve an authentication credential for the specific Client (see Section 4.3.1).
- OPT9: Optionally, define a default group rekeying scheme to refer to in case the 'rekeying_scheme' parameter is not included in the Join Response (see Section 4.3.1).
- OPT10: Optionally, specify the functionalities implemented at the resource hosted by the Client at the URI indicated in the 'control_group_uri' parameter, including the encoding of exchanged messages and other details (see Section 4.3.1).
- OPT11: Optionally, specify policies that instruct Clients to retain messages and for how long, if those are unsuccessfully decrypted (see Section 4.8.1.1). This makes it possible for Clients to decrypt such messages after obtaining updated keying material.

- OPT12: Optionally, specify for the KDC to perform a group rekeying when receiving a Key Renewal Request, together with or instead of renewing individual keying material (see [Section 4.8.2.1](#)).
- OPT13: Optionally, specify how the identifier of a group member's authentication credential is included in requests sent to other group members (see [Section 4.9.1.1](#)).
- OPT14: Optionally, specify additional information to include in rekeying messages for the "Point-to-Point" group rekeying scheme (see [Section 6](#)).

Appendix B. Extensibility for Future COSE Algorithms

As defined in [Section 8.1](#) of [RFC9053], future algorithms can be registered in the "COSE Algorithms" registry [[COSE.Algorithms](#)] as specifying none or multiple COSE capabilities.

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

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

B.1. Format of 'sign_info_entry'

The format of each 'sign_info_entry' (see [Section 3.3.1](#)) is generalized as follows.

- 'sign_parameters' includes $N \geq 0$ elements, each of which is a COSE capability of the signature algorithm indicated in 'sign_alg'.

In particular, 'sign_parameters' has the same format and value of the COSE capabilities array for the signature algorithm indicated in 'sign_alg', as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [[COSE.Algorithms](#)].

- 'sign_key_parameters' is replaced by N elements 'sign_capab', each of which is a CBOR array.

The i -th 'sign_capab' array ($i = 0, \dots, N-1$) is the array of COSE capabilities for the algorithm capability specified in 'sign_parameters'[i].

In particular, each 'sign_capab' array has the same format and value of the COSE capabilities array for the algorithm capability specified in 'sign_parameters'[i].

Such a COSE capabilities array is currently defined for the algorithm capability COSE key type in the "Capabilities" column of the "COSE Key Types" registry [[COSE.Key.Types](#)].

```
sign_info_entry =
[
  id : gname / [+ gname],
  sign_alg : int / tstr,
  sign_parameters : [* alg_capab : any],
  * sign_capab : [* capab : any],
  cred_fmt : int / null
]

gname = tstr
```

Figure 38: 'sign_info_entry' with a General Format

Acknowledgments

The following individuals were helpful in shaping this document: Christian Amsüss, Carsten Bormann, Roman Danyliw, Martin Duke, Thomas Fossati, Vidhi Goel, Rikard Höglund, Ben Kaduk, Erik Kline, Warren Kumari, Watson Ladd, Daniel Migault, John Preuß Mattsson, Zaheduzzaman Sarker, Jim Schaad, Ludwig Seitz, Göran Selander, Cigdem Sengul, Dave Thaler, Henry Thompson, Peter van der Stok, and Paul Wouters.

The work on this document has been partly supported by the Sweden's Innovation Agency 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-164 40 Kista
Sweden
Email: francesca.palombini@ericsson.com

Marco Tiloca

RISE AB
Isafjordsgatan 22
SE-164 40 Kista
Sweden
Email: marco.tiloca@ri.se