

ACE
Internet-Draft
Intended status: Standards Track
Expires: July 9, 2020

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
M. Richardson
SSW
S. Raza
RISE SICS
January 6, 2020

EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-18

Abstract

Enrollment over Secure Transport (EST) is used as a certificate provisioning protocol over HTTPS. Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) for message exchanges. This document defines how to transport EST payloads over secure CoAP (EST-coaps), which allows constrained devices to use existing EST functionality for provisioning certificates.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 9, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Change Log	3
2. Introduction	7
3. Terminology	7
4. DTLS and conformance to RFC7925 profiles	7
5. Protocol Design	10
5.1. Discovery and URIs	10
5.2. Mandatory/optional EST Functions	13
5.3. Payload formats	13
5.4. Message Bindings	15
5.5. CoAP response codes	15
5.6. Message fragmentation	16
5.7. Delayed Responses	17
5.8. Server-side Key Generation	19
6. HTTPS-CoAPS Registrar	21
7. Parameters	23
8. Deployment limitations	23
9. IANA Considerations	24
9.1. Content-Format Registry	24
9.2. Resource Type registry	24
9.3. Well-Known URIs Registry	25
10. Security Considerations	25
10.1. EST server considerations	25
10.2. HTTPS-CoAPS Registrar considerations	27
11. Contributors	28
12. Acknowledgements	28
13. References	28
13.1. Normative References	28
13.2. Informative References	30
Appendix A. EST messages to EST-coaps	32
A.1. cacerts	33
A.2. enroll / reenroll	35
A.3. serverkeygen	37
A.4. csrattrs	39
Appendix B. EST-coaps Block message examples	40
B.1. cacerts	40
B.2. enroll / reenroll	44
Appendix C. Message content breakdown	45
C.1. cacerts	45
C.2. enroll / reenroll	46

C.3. serverkeygen	48
Authors' Addresses	50

1. Change Log

EDNOTE: Remove this section before publication

-18

IESG Reviews fixes.

Removed spurious lines introduced in v-17 due to xml2rfc v3.

-17

v16 remnants by Ben K.

Typos.

-16

Updates to address Yaron S.'s Secdir review.

Updates to address David S.'s Gen-ART review.

-15

Updates to addressed Ben's AD follow up feedback.

-14

Updates to complete Ben's AD review feedback and discussions.

-13

Updates based on AD's review and discussions

Examples redone without password

-12

Updated section 5 based on Esko's comments and nits identified.

Nits and some clarifications for Esko's new review from 5/21/2019.

Nits and some clarifications for Esko's new review from 5/28/2019.

-11

Updated Server-side keygen to simplify motivation and added paragraphs in Security considerations to point out that random numbers are still needed (feedback from Hannes).

-10

Addressed WGLC comments

More consistent request format in the examples.

Explained root resource difference when there is resource discovery

Clarified when the client is supposed to do discovery

Fixed nits and minor Option length inaccuracies in the examples.

-09

WGLC comments taken into account

consensus about discovery of content-format

added additional path for content-format selection

merged DTLS sections

-08

added application/pkix-cert Content-Format TBD287.

discovery text clarified

Removed text on ct negotiation in connection to multipart-core

removed text that duplicates or contradicts RFC7252 (thanks Klaus)

Stated that well-known/est is compulsory

Use of response codes clarified.

removed bugs: Max-Age and Content-Format Options in Request

Accept Option explained for est/skg and added in enroll example

Added second URI /skc for server-side key gen and a simple cert (not PKCS#7)

Persistence of DTLS connection clarified.

Minor text fixes.

-07:

redone examples from scratch with openssl

Updated authors.

Added CoAP RST as a MAY for an equivalent to an HTTP 204 message.

Added serialization example of the /skg CBOR response.

Added text regarding expired IDevIDs and persistent DTLS connection that will start using the Explicit TA Database in the new DTLS connection.

Nits and fixes

Removed CBOR envelop for binary data

Replaced TBD8 with 62.

Added RFC8174 reference and text.

Clarified MTI for server-side key generation and Content-Formats. Defined the /skg MTI (PKCS#8) and the cases where CMS encryption will be used.

Moved Fragmentation section up because it was referenced in sections above it.

-06:

clarified discovery section, by specifying that no discovery may be needed for /.well-known/est URI.

added resource type values for IANA

added list of compulsory to implement and optional functions.

Fixed issues pointed out by the idnits tool.

Updated CoAP response codes section with more mappings between EST HTTP codes and EST-coaps CoAP codes.

Minor updates to the MTI EST Functions section.

Moved Change Log section higher.

-05:

repaired again

TBD8 = 62 removed from C-F registration, to be done in CT draft.

-04:

Updated Delayed response section to reflect short and long delay options.

-03:

Removed observe and simplified long waits

Repaired Content-Format specification

-02:

Added parameter discussion in section 8

Concluded Content-Format specification using multipart-ct draft
examples updated

-01:

Editorials done.

Redefinition of proxy to Registrar in Section 6. Explained better the role of https-coaps Registrar, instead of "proxy"

Provide "observe" Option examples

extended block message example.

inserted new server key generation text in Section 5.8 and motivated server key generation.

Broke down details for DTLS 1.3

New Media-Type uses CBOR array for multiple Content-Format payloads

provided new Content-Format tables

new media format for IANA

-00

copied from vanderstok-ace-coap-04

2. Introduction

"Classical" Enrollment over Secure Transport (EST) [RFC7030] is used for authenticated/authorized endpoint certificate enrollment (and optionally key provisioning) through a Certificate Authority (CA) or Registration Authority (RA). EST transports messages over HTTPS.

This document defines a new transport for EST based on the Constrained Application Protocol (CoAP) since some Internet of Things (IoT) devices use CoAP instead of HTTP. Therefore, this specification utilizes DTLS [RFC6347] and CoAP [RFC7252] instead of TLS [RFC8446] and HTTP [RFC7230].

EST responses can be relatively large and for this reason this specification also uses CoAP Block-Wise Transfer [RFC7959] to offer a fragmentation mechanism of EST messages at the CoAP layer.

This document also profiles the use of EST to only support certificate-based client authentication. HTTP Basic or Digest authentication (as described in Section 3.2.3 of [RFC7030]) are not supported.

3. 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.

Many of the concepts in this document are taken from [RFC7030]. Consequently, much text is directly traceable to [RFC7030].

4. DTLS and conformance to RFC7925 profiles

This section describes how EST-coaps conforms to the profiles of low-resource devices described in [RFC7925]. EST-coaps can transport certificates and private keys. Certificates are responses to (re-)enrollment requests or requests for a trusted certificate list. Private keys can be transported as responses to a server-side key generation request as described in Section 4.4 of [RFC7030] (and subsections) and discussed in Section 5.8 of this document.

EST-coaps depends on a secure transport mechanism that secures the exchanged CoAP messages. DTLS is one such secure protocol. No other changes are necessary regarding the secure transport of EST messages.

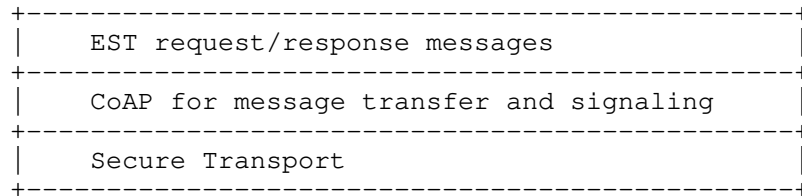


Figure 1: EST-coaps protocol layers

In accordance with sections 3.3 and 4.4 of [RFC7925], the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. Curve secp256r1 MUST be supported [RFC8422]; this curve is equivalent to the NIST P-256 curve. After the publication of [RFC7748], support for Curve25519 will likely be required in the future by (D)TLS Profiles for the Internet of Things [RFC7925].

DTLS 1.2 implementations must use the Supported Elliptic Curves and Supported Point Formats Extensions in [RFC8422]. Uncompressed point format must also be supported. DTLS 1.3 [I-D.ietf-tls-dtls13] implementations differ from DTLS 1.2 because they do not support point format negotiation in favor of a single point format for each curve. Thus, support for DTLS 1.3 does not mandate point format extensions and negotiation. In addition, in DTLS 1.3 the Supported Elliptic Curves extension has been renamed to Supported Groups.

CoAP was designed to avoid IP fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. If fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over several records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

The authentication of the EST-coaps server by the EST-coaps client is based on certificate authentication in the DTLS handshake. The EST-coaps client MUST be configured with at least an Implicit TA database which will enable the authentication of the server the first time before updating its trust anchor (Explicit TA) [RFC7030].

The authentication of the EST-coaps client MUST be with a client certificate in the DTLS handshake. This can either be

- o a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients.
- o a previously installed certificate (e.g., manufacturer IDevID [ieee802.1ar] or a certificate issued by some other party). IDevID's are expected to have a very long life, as long as the device, but under some conditions could expire. In that case, the server MAY authenticate a client certificate against its trust store although the certificate is expired (Section 10).

EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in Section 3 of [RFC7030].

As described in Section 2.1 of [RFC5272] proof-of-identity refers to a value that can be used to prove that an end-entity or client is in the possession of and can use the private key corresponding to the certified public key. Additionally, channel-binding information can link proof-of-identity with an established connection. Connection-based proof-of-possession is OPTIONAL for EST-coaps clients and servers. When proof-of-possession is desired, a set of actions are required regarding the use of tls-unique, described in Section 3.5 in [RFC7030]. The tls-unique information consists of the contents of the first "Finished" message in the (D)TLS handshake between server and client [RFC5929]. The client adds the "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request [RFC5967] to prove that the client is indeed in control of the private key at the time of the (D)TLS session establishment.

In the case of handshake message fragmentation, if proof-of-possession is desired, the Finished message added as the ChallengePassword in the CSR is calculated as specified by the DTLS standards. We summarize it here for convenience. For DTLS 1.2, in the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message must be computed on each reassembled message, as if each message had not been fragmented (Section 4.2.6 of [RFC6347]). The Finished message is calculated as shown in Section 7.4.9 of [RFC5246]. Similarly, for DTLS 1.3, the Finished message must be computed as if each handshake message had been sent as a single fragment (Section 5.8 of [I-D.ietf-tls-dtls13]) following the algorithm described in 4.4.4 of [RFC8446].

In a constrained CoAP environment, endpoints can't always afford to establish a DTLS connection for every EST transaction. An EST-coaps DTLS connection MAY remain open for sequential EST transactions, which was not the case with [RFC7030]. For example, if a /crts request is followed by a /sen request, both can use the same

authenticated DTLS connection. However, when a /crtts request is included in the set of sequential EST transactions, some additional security considerations apply regarding the use of the Implicit and Explicit TA database as explained in Section 10.1.

Given that after a successful enrollment, it is more likely that a new EST transaction will not take place for a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other. These could include a /sen immediately following a /crtts when a device is getting bootstrapped. In some cases, like NAT rebinding, keeping the state of a connection is not possible when devices sleep for extended periods of time. In such occasions, [I-D.ietf-tls-dtls-connection-id] negotiates a connection ID that can eliminate the need for new handshake and its additional cost; or DTLS session resumption provides a less costly alternative than re-doing a full DTLS handshake.

5. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to avoid IP fragmentation. The use of Blocks for the transfer of larger EST messages is specified in Section 5.6. Figure 1 shows the layered EST-coaps architecture.

The EST-coaps protocol design follows closely the EST design. The supported message types in EST-coaps are:

- o CA certificate retrieval needed to receive the complete set of CA certificates.
- o Simple enroll and re-enroll for a CA to sign client identity public key.
- o Certificate Signing Request (CSR) attribute messages that informs the client of the fields to include in a CSR.
- o Server-side key generation messages to provide a client identity private key when the client chooses so.

While [RFC7030] permits a number of the EST functions to be used without authentication, this specification requires that the client MUST be authenticated for all functions.

5.1. Discovery and URIs

EST-coaps is targeted for low-resource networks with small packets. Two types of installations are possible: (1) rigid ones, where the address and the supported functions of the EST server(s) are known,

and (2) a flexible one, where the EST server and its supported functions need to be discovered.

For both types of installations, saving header space is important and short EST-coaps URIs are specified in this document. These URIs are shorter than the ones in [RFC7030]. Two example EST-coaps resource path names are:

```
coaps://example.com:<port>/.well-known/est/<short-est>
coaps://example.com:<port>/.well-known/est/ArbitraryLabel/<short-est>
```

The short-est strings are defined in Table 1. Arbitrary Labels are usually defined and used by EST CAs in order to route client requests to the appropriate certificate profile. Implementers should consider using short labels to minimize transmission overhead.

The EST-coaps server URIs, obtained through discovery of the EST-coaps resource(s) as shown below, are of the form:

```
coaps://example.com:<port>/<root-resource>/<short-est>
coaps://example.com:<port>/<root-resource>/ArbitraryLabel/<short-est>
```

Figure 5 in Section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the EST URI path to the shorter EST-coaps URI path.

EST	EST-coaps
/cacerts	/crt
/simpleenroll	/sen
/simplereenroll	/sren
/serverkeygen	/skg (PKCS#7)
/serverkeygen	/skc (application/pkix-cert)
/csrattrs	/att

Table 1: Short EST-coaps URI path

The /skg message is the EST /serverkeygen equivalent where the client requests a certificate in PKCS#7 format and a private key. If the client prefers a single application/pkix-cert certificate instead of PKCS#7, it will make an /skc request. In both cases (i.e., /skg, /skc) a private key MUST be returned.

Clients and servers MUST support the short resource EST-coaps URIs.

In the context of CoAP, the presence and location of (path to) the EST resources are discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"ace.est*"` [RFC6690]. The example below shows the discovery over CoAPS of the presence and location of EST-coaps resources. Linefeeds are included only for readability.

```
REQ: GET /.well-known/core?rt=ace.est*
```

```
RES: 2.05 Content
</est/crts>;rt="ace.est.crts";ct="281 TBD287",
</est/sen>;rt="ace.est.sen";ct="281 TBD287",
</est/sren>;rt="ace.est.sren";ct="281 TBD287",
</est/att>;rt="ace.est.att";ct=285,
</est/skg>;rt="ace.est.skg";ct=62,
</est/skc>;rt="ace.est.skc";ct=62
```

The first three lines, describing `ace.est.crts`, `ace.est.sen`, and `ace.est.sren`, of the discovery response above **MUST** be returned if the server supports resource discovery. The last three lines are only included if the corresponding EST functions are implemented (see Table 2). The Content-Formats in the response allow the client to request one that is supported by the server. These are the values that would be sent in the client request with an Accept option.

Discoverable port numbers can be returned in the response payload. An example response payload for non-default CoAPS server port 61617 follows below. Linefeeds are included only for readability.

```
REQ: GET /.well-known/core?rt=ace.est*
```

```
RES: 2.05 Content
<coaps://[2001:db8:3::123]:61617/est/crts>;rt="ace.est.crts";
    ct="281 TBD287",
<coaps://[2001:db8:3::123]:61617/est/sen>;rt="ace.est.sen";
    ct="281 TBD287",
<coaps://[2001:db8:3::123]:61617/est/sren>;rt="ace.est.sren";
    ct="281 TBD287",
<coaps://[2001:db8:3::123]:61617/est/att>;rt="ace.est.att";
    ct=285,
<coaps://[2001:db8:3::123]:61617/est/skg>;rt="ace.est.skg";
    ct=62,
<coaps://[2001:db8:3::123]:61617/est/skc>;rt="ace.est.skc";
    ct=62
```

The server **MUST** support the default `/.well-known/est` root resource. The server **SHOULD** support resource discovery when it supports non-default URIs (like `/est` or `/est/ArbitraryLabel`) or ports. The client

SHOULD use resource discovery when it is unaware of the available EST-coaps resources.

Throughout this document the example root resource of /est is used.

5.2. Mandatory/optional EST Functions

This specification contains a set of required-to-implement functions, optional functions, and not specified functions. The unspecified functions are deemed too expensive for low-resource devices in payload and calculation times.

Table 2 specifies the mandatory-to-implement or optional implementation of the EST-coaps functions. Discovery of the existence of optional functions is described in Section 5.1.

EST Functions	EST-coaps implementation
/cacerts	MUST
/simpleenroll	MUST
/simplereenroll	MUST
/fullcmc	Not specified
/serverkeygen	OPTIONAL
/csrattrs	OPTIONAL

Table 2: List of EST-coaps functions

5.3. Payload formats

EST-coaps is designed for low-resource devices and hence does not need to send Base64-encoded data. Simple binary is more efficient (30% smaller payload for DER-encoded ASN.1) and well supported by CoAP. Thus, the payload for a given Media-Type follows the ASN.1 structure of the Media-Type and is transported in binary format.

The Content-Format (HTTP Content-Type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The Media-Types specified in the HTTP Content-Type header field (Section 3.2.2 of [RFC7030]) are specified by the Content-Format Option (12) of CoAP. The combination of URI-Path and Content-Format in EST-coaps MUST map to an allowed combination of URI and Media-Type in EST. The required Content-Formats for these requests and response messages are defined in Section 9.1. The CoAP response codes are defined in Section 5.5.

Content-Format TBD287 can be used in place of 281 to carry a single certificate instead of a PKCS#7 container in a /crt, /sen, /sren or /skg response. Content-Format 281 MUST be supported by EST-coaps servers. Servers MAY also support Content-Format TBD287. It is up to the client to support only Content-Format 281, TBD287 or both. The client will use a COAP Accept Option in the request to express the preferred response Content-Format. If an Accept Option is not included in the request, the client is not expressing any preference and the server SHOULD choose format 281.

Content-Format 286 is used in /sen, /sren and /skg requests and 285 in /att responses.

A representation with Content-Format identifier 62 contains a collection of representations along with their respective Content-Format. The Content-Format identifies the Media-Type application/multipart-core specified in [I-D.ietf-core-multipart-ct]. For example, a collection, containing two representations in response to a EST-coaps server-side key generation /skg request, could include a private key in PKCS#8 [RFC5958] with Content-Format identifier 284 (0x011C) and a single certificate in a PKCS#7 container with Content-Format identifier 281 (0x0119). Such a collection would look like [284,h'0123456789abcdef', 281,h'fedcba9876543210'] in diagnostic CBOR notation. The serialization of such CBOR content would be

```

84          # array(4)
19 011C     # unsigned(284)
48          # bytes(8)
  0123456789ABCDEF # "\x01#Eg\x89\xAB\xCD\xEF"
19 0119     # unsigned(281)
48          # bytes(8)
  FEDCBA9876543210 # "\xFE\xDC\xBA\x98vT2\x10"
```

Multipart /skg response serialization

When the client makes an /skc request the certificate returned with the private key is a single X.509 certificate (not a PKCS#7 container) with Content-Format identifier TBD287 (0x011F) instead of 281. In cases where the private key is encrypted with CMS (as explained in Section 5.8) the Content-Format identifier is 280 (0x0118) instead of 284. The content format used in the response is summarized in Table 3.

Function	Response part 1	Response part 2
/skg	284	281
/skc	280	TBD287

Table 3: response content formats for skg and skc

The key and certificate representations are DER-encoded ASN.1, in its native binary form. An example is shown in Appendix A.3.

5.4. Message Bindings

The general EST-coaps message characteristics are:

- o EST-coaps servers sometimes need to provide delayed responses which are preceded by an immediately returned empty ACK or an ACK containing response code 5.03 as explained in Section 5.7. Thus, it is RECOMMENDED for implementers to send EST-coaps requests in confirmable CON CoAP messages.
- o The CoAP Options used are Uri-Host, Uri-Path, Uri-Port, Content-Format, Block1, Block2, and Accept. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages. The Uri-host and Uri-Port Options can be omitted from the COAP message sent on the wire. When omitted, they are logically assumed to be the transport protocol destination address and port respectively. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly. Other COAP Options should be handled in accordance with [RFC7252].
- o EST URLs are HTTPS based (https://), in CoAP these are assumed to be translated to CoAPS (coaps://)

Table 1 provides the mapping from the EST URI path to the EST-coaps URI path. Appendix A includes some practical examples of EST messages translated to CoAP.

5.5. CoAP response codes

Section 5.9 of [RFC7252] and Section 7 of [RFC8075] specify the mapping of HTTP response codes to CoAP response codes. The success code in response to an EST-coaps GET request (/crt, /att), is 2.05. Similarly, 2.04 is used in successful response to EST-coaps POST requests (/sen, /sren, /skg, /skc).

EST makes use of HTTP 204 or 404 responses when a resource is not available for the client. In EST-coaps 2.04 is used in response to a POST (/sen, /sren, /skg, /skc). 4.04 is used when the resource is not available for the client.

HTTP response code 202 with a Retry-After header field in [RFC7030] has no equivalent in CoAP. HTTP 202 with Retry-After is used in EST for delayed server responses. Section 5.7 specifies how EST-coaps handles delayed messages with 5.03 responses with a Max-Age Option.

Additionally, EST's HTTP 400, 401, 403, 404 and 503 status codes have their equivalent CoAP 4.00, 4.01, 4.03, 4.04 and 5.03 response codes in EST-coaps. Table 4 summarizes the EST-coaps response codes.

operation	EST-coaps response code	Description
/crt, /att	2.05	Success. Certs included in the response payload.
/sen, /skg, /sren, /skc	4.xx / 5.xx	Failure.
	2.04	Success. Cert included in the response payload.
	5.03	Retry in Max-Age Option time.
	4.xx / 5.xx	Failure.

Table 4: EST-coaps response codes

5.6. Message fragmentation

DTLS defines fragmentation only for the handshake and not for secure data exchange (DTLS records). [RFC6347] states that to avoid using IP fragmentation, which involves error-prone datagram reconstitution, invokers of the DTLS record layer should size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 [ieee802.15.4] network are recommended to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames.

That is not always possible in EST-coaps. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and Object Identifier (OID) fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, Subject Alternative Names (SAN) and cert fields. For 384-bit curves, ECDSA certificates increase in size and can sometimes reach 1.5KB.

Additionally, there are times when the EST certificate response from the server can include multiple certificates that amount to large payloads. Section 4.6 of CoAP [RFC7252] describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Section 4.6 of [RFC7252] also suggests that IPv4 implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes. Even with ECC, EST-coaps messages can still exceed MTU sizes on the Internet or 6LoWPAN [RFC4919] (Section 2 of [RFC7959]). EST-coaps needs to be able to fragment messages into multiple DTLS datagrams.

To perform fragmentation in CoAP, [RFC7959] specifies the Block1 Option for fragmentation of the request payload and the Block2 Option for fragmentation of the return payload of a CoAP flow. As explained in Section 1 of [RFC7959], block-wise transfers should be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks. EST-coaps servers MUST implement Block1 and Block2. EST-coaps clients MUST implement Block2. EST-coaps clients MUST implement Block1 only if they are expecting to send EST-coaps requests with a packet size that exceeds the Path MTU.

[RFC7959] also defines Size1 and Size2 Options to provide size information about the resource representation in a request and response. EST-client and server MAY support Size1 and Size2 Options.

Examples of fragmented EST-coaps messages are shown in Appendix B.

5.7. Delayed Responses

Server responses can sometimes be delayed. According to Section 5.2.2 of [RFC7252], a slow server can acknowledge the request and respond later with the requested resource representation. In particular, a slow server can respond to an EST-coaps enrollment request with an empty ACK with code 0.00, before sending the certificate to the client after a short delay. If the certificate response is large, the server will need more than one Block2 block to transfer it.

This situation is shown in Figure 2. The client sends an enrollment request that uses $N1+1$ Block1 blocks. The server uses an empty 0.00 ACK to announce the delayed response which is provided later with 2.04 messages containing $N2+1$ Block2 Options. The first 2.04 is a confirmable message that is acknowledged by the client. Onwards, the client acknowledges all subsequent Block2 blocks. The notation of Figure 2 is explained in Appendix B.1.

```

POST [2001:db8::2:1]:61616/est/sen (CON) (1:0/1/256) {CSR (frag# 1)} -->
  <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON) (1:1/1/256) {CSR (frag# 2)} -->
  <-- (ACK) (1:1/1/256) (2.31 Continue)
      .
      .
      .
POST [2001:db8::2:1]:61616/est/sen (CON) (1:N1/0/256) {CSR (frag# N1+1)} -->
  <-- (0.00 empty ACK)
      |
  ... Short delay before the certificate is ready ...
      |
  <-- (CON) (1:N1/0/256) (2:0/1/256) (2.04 Changed) {Cert resp (frag# 1)}
                                     (ACK)                                -->
POST [2001:db8::2:1]:61616/est/sen (CON) (2:1/0/256)                                -->
  <-- (ACK) (2:1/1/256) (2.04 Changed) {Cert resp (frag# 2)}
      .
      .
      .
POST [2001:db8::2:1]:61616/est/sen (CON) (2:N2/0/256)                                -->
  <-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 2: EST-COAP enrollment with short wait

If the server is very slow (for example, manual intervention is required which would take minutes), it SHOULD respond with an ACK containing response code 5.03 (Service unavailable) and a Max-Age Option to indicate the time the client SHOULD wait before sending another request to obtain the content. After a delay of Max-Age, the client SHOULD resend the identical CSR to the server. As long as the server continues to respond with response code 5.03 (Service Unavailable) with a Max-Age Option, the client will continue to delay for Max-Age and then resend the enrollment request until the server responds with the certificate or the client abandons the request for policy or other reasons.

To demonstrate this scenario, Figure 3 shows a client sending an enrollment request that uses N1+1 Block1 blocks to send the CSR to the server. The server needs N2+1 Block2 blocks to respond, but also needs to take a long delay (minutes) to provide the response. Consequently, the server uses a 5.03 ACK response with a Max-Age Option. The client waits for a period of Max-Age as many times as it receives the same 5.03 response and retransmits the enrollment request until it receives a certificate in a fragmented 2.04 response.

```

POST [2001:db8::2:1]:61616/est/sen (CON) (1:0/1/256) {CSR (frag# 1)} -->
<-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON) (1:1/1/256) {CSR (frag# 2)} -->
<-- (ACK) (1:1/1/256) (2.31 Continue)
.
.
.
POST [2001:db8::2:1]:61616/est/sen (CON) (1:N1/0/256) {CSR (frag# N1+1)} -->
<-- (ACK) (1:N1/0/256) (5.03 Service Unavailable) (Max-Age)
|
... Client tries again after Max-Age with identical payload ...
|
POST [2001:db8::2:1]:61616/est/sen (CON) (1:0/1/256) {CSR (frag# 1)} -->
<-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON) (1:1/1/256) {CSR (frag# 2)} -->
<-- (ACK) (1:1/1/256) (2.31 Continue)
.
.
.
POST [2001:db8::2:1]:61616/est/sen (CON) (1:N1/0/256) {CSR (frag# N1+1)} -->
|
... Immediate response when certificate is ready ...
|
<-- (ACK) (1:N1/0/256) (2:0/1/256) (2.04 Changed) {Cert resp (frag# 1)}
POST [2001:db8::2:1]:61616/est/sen (CON) (2:1/0/256) -->
<-- (ACK) (2:1/1/256) (2.04 Changed) {Cert resp (frag# 2)}
.
.
.
POST [2001:db8::2:1]:61616/est/sen (CON) (2:N2/0/256) -->
<-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 3: EST-COAP enrollment with long wait

5.8. Server-side Key Generation

Private keys can be generated on the server to support scenarios where server-side key generation is needed. Such scenarios include those where it is considered more secure to generate the long-lived, random private key that identifies the client at the server, or where the resources spent to generate a random private key at the client are considered scarce, or where the security policy requires that the certificate public and corresponding private keys are centrally generated and controlled. As always, it is necessary to use proper random numbers in various protocols such as (D)TLS (Section 10.1).

When requesting server-side key generation, the client asks for the server or proxy to generate the private key and the certificate, which are transferred back to the client in the server-side key generation response. In all respects, the server treats the CSR as it would treat any enroll or re-enroll CSR; the only distinction here is that the server MUST ignore the public key values and signature in the CSR. These are included in the request only to allow re-use of existing codebases for generating and parsing such requests.

The client /skg request is for a certificate in a PKCS#7 container and private key in two application/multipart-core elements. Respectively, an /skc request is for a single application/pkix-cert certificate and a private key. The private key Content-Format requested by the client is indicated in the PKCS#10 CSR request. If the request contains SMIMECapabilities and DecryptKeyIdentifier or AsymmetricDecryptKeyIdentifier the client is expecting Content-Format 280 for the private key. Then this private key is encrypted symmetrically or asymmetrically as per [RFC7030]. The symmetric key or the asymmetric keypair establishment method is out of scope of this specification. A /skg or /skc request with a CSR without SMIMECapabilities expects an application/multipart-core with an unencrypted PKCS#8 private key with Content-Format 284.

The EST-coaps server-side key generation response is returned with Content-Format application/multipart-core [I-D.ietf-core-multipart-ct] containing a CBOR array with four items (Section 5.3). The two representations (each consisting of two CBOR array items) do not have to be in a particular order since each representation is preceded by its Content-Format ID. Depending on the request, the private key can be in unprotected PKCS#8 [RFC5958] format (Content-Format 284) or protected inside of CMS SignedData (Content-Format 280). The SignedData, placed in the outermost container, is signed by the party that generated the private key, which may be the EST server or the EST CA. SignedData placed within the Enveloped Data does not need additional signing as explained in Section 4.4.2 of [RFC7030]. In summary, the symmetrically encrypted key is included in the encryptedKey attribute in a KEKRecipientInfo structure. In the case where the asymmetric encryption key is suitable for transport key operations the generated private key is encrypted with a symmetric key. The symmetric key itself is encrypted by the client-defined (in the CSR) asymmetric public key and is carried in an encryptedKey attribute in a KeyTransRecipientInfo structure. Finally, if the asymmetric encryption key is suitable for key agreement, the generated private key is encrypted with a symmetric key. The symmetric key itself is encrypted by the client defined (in the CSR) asymmetric public key and is carried in an recipientEncryptedKeys attribute in a KeyAgreeRecipientInfo.

[RFC7030] recommends the use of additional encryption of the returned private key. For the context of this specification, clients and servers that choose to support server-side key generation **MUST** support unprotected (PKCS#8) private keys (Content-Format 284). Symmetric or asymmetric encryption of the private key (CMS EnvelopedData, Content-Format 280) **SHOULD** be supported for deployments where end-to-end encryption is needed between the client and a server. Such cases could include architectures where an entity between the client and the CA terminates the DTLS connection (Registrar in Figure 4). Although [RFC7030] strongly recommends that clients request the use of CMS encryption on top of the TLS channel's protection, this document does not make such a recommendation; CMS encryption can still be used when mandated by the use-case.

6. HTTPS-CoAPS Registrar

In real-world deployments, the EST server will not always reside within the CoAP boundary. The EST server can exist outside the constrained network in which case it will support TLS/HTTP instead of CoAPS. In such environments EST-coaps is used by the client within the CoAP boundary and TLS is used to transport the EST messages outside the CoAP boundary. A Registrar at the edge is required to operate between the CoAP environment and the external HTTP network as shown in Figure 4.

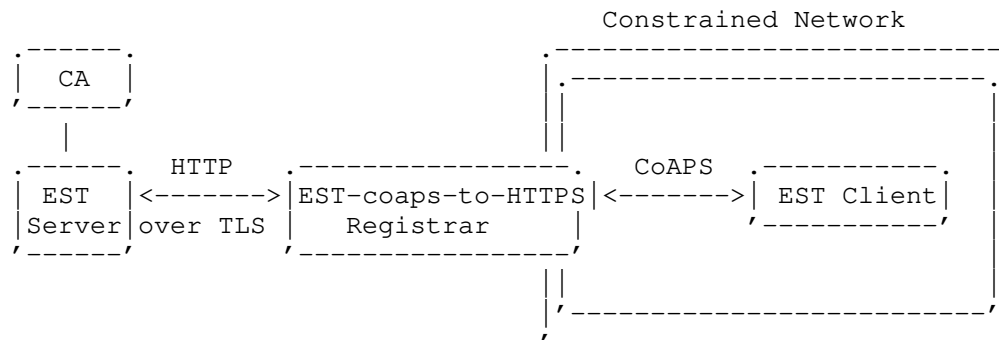


Figure 4: EST-coaps-to-HTTPS Registrar at the CoAP boundary.

The EST-coaps-to-HTTPS Registrar **MUST** terminate EST-coaps downstream and initiate EST connections over TLS upstream. The Registrar **MUST** authenticate and optionally authorize the client requests while it **MUST** be authenticated by the EST server or CA. The trust relationship between the Registrar and the EST server **SHOULD** be pre-established for the Registrar to proxy these connections on behalf of various clients.

When enforcing Proof-of-Possession (PoP) linking, the DTLS tls-unique value of the (D)TLS session is used to prove that the private key corresponding to the public key is in the possession of the client and was used to establish the connection as explained in Section 4. The PoP linking information is lost between the EST-coaps client and the EST server when a Registrar is present. The EST server becomes aware of the presence of a Registrar from its TLS client certificate that includes id-kp-cmcRA [RFC6402] extended key usage extension (EKU). As explained in Section 3.7 of [RFC7030], the "EST server SHOULD apply an authorization policy consistent with a Registrar client. For example, it could be configured to accept PoP linking information that does not match the current TLS session because the authenticated EST client Registrar has verified this information when acting as an EST server".

Table 1 contains the URI mappings between EST-coaps and EST that the Registrar MUST adhere to. Section 5.5 of this specification and Section 7 of [RFC8075] define the mappings between EST-coaps and HTTP response codes, that determine how the Registrar MUST translate CoAP response codes from/to HTTP status codes. The mapping from CoAP Content-Format to HTTP Content-Type is defined in Section 9.1. Additionally, a conversion from CBOR major type 2 to Base64 encoding MUST take place at the Registrar. If CMS end-to-end encryption is employed for the private key, the encrypted CMS EnvelopedData blob MUST be converted at the Registrar to binary CBOR type 2 downstream to the client. This is a format conversion that does not require decryption of the CMS EnvelopedData.

A deviation from the mappings in Table 1 could take place if clients that leverage server-side key generation preferred for the enrolled keys to be generated by the Registrar in the case the CA does not support server-side key generation. Such a Registrar is responsible for generating a new CSR signed by a new key which will be returned to the client along with the certificate from the CA. In these cases, the Registrar MUST use random number generation with proper entropy.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP Registrar MUST reassemble the BLOCKs before translating the binary content to Base64, and consecutively relay the message upstream.

The EST-coaps-to-HTTP Registrar MUST support resource discovery according to the rules in Section 5.1.

7. Parameters

This section addresses transmission parameters described in sections 4.7 and 4.8 of [RFC7252]. EST does not impose any unique values on the CoAP parameters in [RFC7252], but the setting of the CoAP parameter values may have consequence for the setting of the EST parameter values.

Implementations should follow the default CoAP configuration parameters [RFC7252]. However, depending on the implementation scenario, retransmissions and timeouts can also occur on other networking layers, governed by other configuration parameters. When a change in a server parameter has taken place, the parameter values in the communicating endpoints MUST be adjusted as necessary. Examples of how parameters could be adjusted include higher layer congestion protocols, provisioning agents and configurations included in firmware updates.

Some further comments about some specific parameters, mainly from Table 2 in [RFC7252]:

- o NSTART: A parameter that controls the number of simultaneous outstanding interactions that a client maintains to a given server. An EST-coaps client is expected to control at most one interaction with a given server, which is the default NSTART value defined in [RFC7252].
- o DEFAULT_LEISURE: This setting is only relevant in multicast scenarios, outside the scope of EST-coaps.
- o PROBING_RATE: A parameter which specifies the rate of re-sending non-confirmable messages. In the rare situations that non-confirmable messages are used, the default PROBING_RATE value defined in [RFC7252] applies.

Finally, the Table 3 parameters in [RFC7252] are mainly derived from Table 2. Directly changing parameters on one table would affect parameters on the other.

8. Deployment limitations

Although EST-coaps paves the way for the utilization of EST by constrained devices in constrained networks, some classes of devices [RFC7228] will not have enough resources to handle the payloads that come with EST-coaps. The specification of EST-coaps is intended to ensure that EST works for networks of constrained devices that choose to limit their communications stack to DTLS/CoAP. It is up to the

network designer to decide which devices execute the EST protocol and which do not.

9. IANA Considerations

9.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry [COREparams] are specified in Table 5. These have been registered provisionally in the IETF Review or IESG Approval range (256-9999).

HTTP Content-Type	ID	Reference
application/pkcs7-mime; smime-type=server-generated-key	280	[RFC7030] [I-D.ietf-lamps-rfc5751-bis] [ThisRFC]
application/pkcs7-mime; smime-type=certs-only	281	[I-D.ietf-lamps-rfc5751-bis] [ThisRFC]
application/pkcs8	284	[RFC5958] [I-D.ietf-lamps-rfc5751-bis] [ThisRFC]
application/csrattrs	285	[RFC7030]
application/pkcs10	286	[RFC5967] [I-D.ietf-lamps-rfc5751-bis] [ThisRFC]
application/pkix-cert	TBD287	[RFC2585] [ThisRFC]

Table 5: New CoAP Content-Formats

It is suggested that 287 is allocated to TBD287.

9.2. Resource Type registry

This memo registers new Resource Type (rt=) Link Target Attributes in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

- o rt="ace.est.certs". This resource depicts the support of EST get cacerts.
- o rt="ace.est.sen". This resource depicts the support of EST simple enroll.
- o rt="ace.est.sren". This resource depicts the support of EST simple reenroll.

- o `rt="ace.est.att"`. This resource depicts the support of EST get CSR attributes.
- o `rt="ace.est.skg"`. This resource depicts the support of EST server-side key generation with the returned certificate in a PKCS#7 container.
- o `rt="ace.est.skc"`. This resource depicts the support of EST server-side key generation with the returned certificate in application/pkix-cert format.

9.3. Well-Known URIs Registry

A new additional reference is requested for the est URI in the Well-Known URIs registry:

URI Suffix	Change Controller	References	Status	Related Information	Date Registered	Date Modified
est	IETF	[RFC7030] [THIS RFC]	permanent		2013-08-16	[THIS RFC's publication date]

10. Security Considerations

10.1. EST server considerations

The security considerations of Section 6 of [RFC7030] are only partially valid for the purposes of this document. As HTTP Basic Authentication is not supported, the considerations expressed for using passwords do not apply. The other portions of the security considerations of [RFC7030] continue to apply.

Modern security protocols require random numbers to be available during the protocol run, for example for nonces and ephemeral (EC) Diffie-Hellman key generation. This capability to generate random numbers is also needed when the constrained device generates the private key (that corresponds to the public key enrolled in the CSR). When server-side key generation is used, the constrained device depends on the server to generate the private key randomly, but it still needs locally generated random numbers for use in security protocols, as explained in Section 12 of [RFC7925]. Additionally, the transport of keys generated at the server is inherently risky. For those deploying server-side key generation, analysis SHOULD be

done to establish whether server-side key generation increases or decreases the probability of digital identity theft.

It is important to note that, as pointed out in [PsQs], sources contributing to the randomness pool used to generate random numbers on laptops or desktop PCs, such as mouse movement, timing of keystrokes, or air turbulence on the movement of hard drive heads, are not available on many constrained devices. Other sources have to be used or dedicated hardware has to be added. Selecting hardware for an IoT device that is capable of producing high-quality random numbers is therefore important [RSAfact].

As discussed in Section 6 of [RFC7030], it is "RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication is carefully managed to reduce the chance of a third-party CA with poor certification practices jeopardizing authentication. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3) limits any risk to the first TLS exchange". Alternatively, in a case where a /sen request immediately follows a /crts, a client MAY choose to keep the connection authenticated by the Implicit TA open for efficiency reasons (Section 4). A client that interleaves EST-coaps /crts request with other requests in the same DTLS connection SHOULD revalidate the server certificate chain against the updated Explicit TA from the /crts response before proceeding with the subsequent requests. If the server certificate chain does not authenticate against the database, the client SHOULD close the connection without completing the rest of the requests. The updated Explicit TA MUST continue to be used in new DTLS connections.

In cases where the IDevID used to authenticate the client is expired the server MAY still authenticate the client because IDevIDs are expected to live as long as the device itself (Section 4). In such occasions, checking the certificate revocation status or authorizing the client using another method is important for the server to raise its confidence that the client can be trusted.

In accordance with [RFC7030], TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More recommendations for secure use of TLS and DTLS are included in [BCP195].

As described in CMC, Section 6.7 of [RFC5272], "For keys that can be used as signature keys, signing the certification request with the private key serves as a PoP on that key pair". The inclusion of tls-unique in the certificate request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that only the authenticated client currently has access to the private key.

What's more, CMC PoP linking uses `tls-unique` as it is defined in [RFC5929]. The 3SHAKE attack [triplesshake] poses a risk by allowing a man-in-the-middle to leverage session resumption and renegotiation to inject himself between a client and server even when channel binding is in use. Implementers should use the Extended Master Secret Extension in DTLS [RFC7627] to prevent such attacks. In the context of this specification, an attacker could invalidate the purpose of the PoP linking ChallengePassword in the client request by resuming an EST-coaps connection. Even though the practical risk of such an attack to EST-coaps is not devastating, we would rather use a more secure channel binding mechanism. Such a mechanism could include an updated `tls-unique` value generation like the `tls-unique-prf` defined in [I-D.josefsson-sasl-tls-cb] by using a TLS exporter [RFC5705] in TLS 1.2 or TLS 1.3's updated exporter (Section 7.5 of [RFC8446]) value in place of the `tls-unique` value in the CSR. Such mechanism has not been standardized yet. Adopting a channel binding value generated from an exporter would break backwards compatibility for an RA that proxies through to a classic EST server. Thus, in this specification we still depend on the `tls-unique` mechanism defined in [RFC5929], especially since a 3SHAKE attack does not expose messages exchanged with EST-coaps.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

10.2. HTTPS-CoAPS Registrar considerations

The Registrar proposed in Section 6 must be deployed with care, and only when direct client-server connections are not possible. When PoP linking is used the Registrar terminating the DTLS connection establishes a new TLS connection with the upstream CA. Thus, it is impossible for PoP linking to be enforced end-to-end for the EST transaction. The EST server could be configured to accept PoP linking information that does not match the current TLS session because the authenticated EST Registrar is assumed to have verified PoP linking downstream to the client.

The introduction of an EST-coaps-to-HTTP Registrar assumes the client can authenticate the Registrar using its implicit or explicit TA database. It also assumes the Registrar has a trust relationship with the upstream EST server in order to act on behalf of the clients. When a client uses the Implicit TA database for certificate validation, it SHOULD confirm if the server is acting as an RA by the presence of the `id-kp-cmcRA` EKU [RFC6402] in the server certificate.

In a server-side key generation case, if no end-to-end encryption is used, the Registrar may be able to see the private key as it acts as a man-in-the-middle. Thus, the client puts its trust on the Registrar not exposing the private key.

Clients that leverage server-side key generation without end-to-end encryption of the private key (Section 5.8) have no knowledge if the Registrar will be generating the private key and enrolling the certificates with the CA or if the CA will be responsible for generating the key. In such cases, the existence of a Registrar requires the client to put its trust on the registrar when it is generating the private key.

11. Contributors

Martin Furuheid contributed to the EST-coaps specification by providing feedback based on the Nexus EST over CoAPS server implementation that started in 2015. Sandeep Kumar kick-started this specification and was instrumental in drawing attention to the importance of the subject.

12. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLs and his support for the Content-Format specification. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution. They would also like to thank Peter Panburana for his feedback on technical details of the solution. Constructive comments were received from Benjamin Kaduk, Eliot Lear, Jim Schaad, Hannes Tschofenig, Julien Vermillard, John Manuel, Oliver Pfaff, Pete Beal and Carsten Bormann.

Interop tests were done by Oliver Pfaff, Thomas Werner, Oskar Camezind, Bjorn Elmers and Joel Hoglund.

Robert Moskowitz provided code to create the examples.

13. References

13.1. Normative References

- [I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-04 (work in progress), August 2019.

- [I-D.ietf-lamps-rfc5751-bis]
Schaad, J., Ramsdell, B., and S. Turner, "Secure/
Multipurpose Internet Mail Extensions (S/MIME) Version 4.0
Message Specification", draft-ietf-lamps-rfc5751-bis-12
(work in progress), September 2018.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The
Datagram Transport Layer Security (DTLS) Protocol Version
1.3", draft-ietf-tls-dtls13-34 (work in progress),
November 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key
Infrastructure Operational Protocols: FTP and HTTP",
RFC 2585, DOI 10.17487/RFC2585, May 1999,
<<https://www.rfc-editor.org/info/rfc2585>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246,
DOI 10.17487/RFC5246, August 2008,
<<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958,
DOI 10.17487/RFC5958, August 2010,
<<https://www.rfc-editor.org/info/rfc5958>>.
- [RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967,
DOI 10.17487/RFC5967, August 2010,
<<https://www.rfc-editor.org/info/rfc5967>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,
January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
Format", RFC 6690, DOI 10.17487/RFC6690, August 2012,
<<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<https://www.rfc-editor.org/info/rfc7030>>.

- [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>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [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>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [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>>.
- [RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

13.2. Informative References

- [BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <<https://www.rfc-editor.org/info/bcp195>>.
- [COREparams] "Constrained RESTful Environments (CoRE) Parameters", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>>.

- [I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-07 (work in progress), October 2019.
- [I-D.josefsson-sasl-tls-cb]
Josefsson, S., "Channel Bindings for TLS based on the PRF", draft-josefsson-sasl-tls-cb-03 (work in progress), March 2015.
- [I-D.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", draft-moskowitz-ecdsa-pki-07 (work in progress), August 2019.
- [ieee802.15.4]
"IEEE Standard 802.15.4-2006", 2006.
- [ieee802.1ar]
"IEEE 802.1AR Secure Device Identifier", December 2009.
- [PsQs]
"Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", USENIX Security Symposium 2012 ISBN 978-931971-95-9, August 2012.
- [RFC4919]
Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <<https://www.rfc-editor.org/info/rfc4919>>.
- [RFC5272]
Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [RFC5705]
Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC5929]
Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC6402]
Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <<https://www.rfc-editor.org/info/rfc6402>>.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <<https://www.rfc-editor.org/info/rfc7251>>.
- [RFC7299] Housley, R., "Object Identifier Registry for the PKIX Working Group", RFC 7299, DOI 10.17487/RFC7299, July 2014, <<https://www.rfc-editor.org/info/rfc7299>>.
- [RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <<https://www.rfc-editor.org/info/rfc7627>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RSAfact] "Factoring RSA keys from certified smart cards: Coppersmith in the wild", Advances in Cryptology - ASIACRYPT 2013, August 2013.
- [tripleshake]
"Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS", IEEE Security and Privacy ISBN 978-1-4799-4686-0, May 2014.

Appendix A. EST messages to EST-coaps

This section shows similar examples to the ones presented in Appendix A of [RFC7030]. The payloads in the examples are the hex encoded binary, generated with 'xxd -p', of the PKI certificates created following [I-D.moskowitz-ecdsa-pki]. Hex is used for visualization purposes because a binary representation cannot be rendered well in text. The hexadecimal representations would not be transported in hex, but in binary. The payloads are shown

unencrypted. In practice the message content would be transferred over an encrypted DTLS channel.

The certificate responses included in the examples contain Content-Format 281 (application/pkcs7). If the client had requested Content-Format TBD287 (application/pkix-cert) by querying /est/skc, the server would respond with a single DER binary certificate in the multipart-core container.

These examples assume a short resource path of "/est". Even though omitted from the examples for brevity, before making the EST-coaps requests, a client would learn about the server supported EST-coaps resources with a GET request for /.well-known/core?rt=ace.est* as explained in Section 5.1.

The corresponding CoAP headers are only shown in Appendix A.1. Creating CoAP headers is assumed to be generally understood.

The message content breakdown is presented in Appendix C.

A.1. cacerts

In EST-coaps, a cacerts message can be:

```
GET example.com:9085/est/crts
(Accept: 281)
```

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in Appendix B.

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Token = 0x9a (client generated)
Options
Option (Uri-Host)
  Option Delta = 0x3 (option# 3)
  Option Length = 0xB
  Option Value = "example.com"
Option (Uri-Port)
  Option Delta = 0x4 (option# 3+4=7)
  Option Length = 0x2
  Option Value = 9085
Option (Uri-Path)
  Option Delta = 0x4 (option# 7+4=11)
  Option Length = 0x3
  Option Value = "est"
Option (Uri-Path)
  Option Delta = 0x0 (option# 11+0=11)
  Option Length = 0x4
  Option Value = "crts"
Option (Accept)
  Option Delta = 0x6 (option# 11+6=17)
  Option Length = 0x2
  Option Value = 281
Payload = [Empty]
```

As specified in Section 5.10.1 of [RFC7252], the Uri-Host and Uri-Port Options can be omitted if they coincide with the transport protocol destination address and port respectively.

A 2.05 Content response with a cert in EST-coaps will then be

```
2.05 Content (Content-Format: 281)
  {payload with certificate in binary format}
```

with CoAP fields

```
Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a (copied from request by server)
Options
  Option (Content-Format)
    Option Delta = 0xC (option# 12)
    Option Length = 0x2
    Option Value = 281
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
3082027a06092a864886f70d010702a082026b308202670201013100300b
06092a864886f70d010701a082024d30820249308201efa0030201020208
0b8bb0fe604f6a1e300a06082a8648ce3d0403023067310b300906035504
0613025553310b300906035504080c024341310b300906035504070c024c
4131143012060355040a0c0b4578616d706c6520496e6331163014060355
040b0c0d63657274696669636174696f6e3110300e06035504030c07526f
6f74204341301e170d3139303133313131323730335a170d333930313236
3131323730335a3067310b3009060355040613025553310b300906035504
080c024341310b300906035504070c024c4131143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3110300e06035504030c07526f6f742043413059301306072a86
48ce3d020106082a8648ce3d030107034200040c1b1e82ba8cc72680973f
97edb8a0c72ab0d405f05d4fe29b997a14ccce89008313d09666b6ce375c
595fcc8e37f8e4354497011be90e56794bd91ad951ab45a3818430818130
1d0603551d0e041604141df1208944d77b5f1d9dcb51ee244a523f3ef5de
301f0603551d230418301680141df1208944d77b5f1d9dcb51ee244a523f
3ef5de300f0603551d130101ff040530030101ff300e0603551d0f0101ff
040403020106301e0603551d110417301581136365727469667940657861
6d706c652e636f6d300a06082a8648ce3d040302034800304502202b891d
d411d07a6d6f621947635ba4c43165296b3f633726f02e51ecf464bd4002
2100b4be8a80d08675f041fbc719acf3b39dedc85dc92b3035868cb2daa8
f05db196a1003100
```

The breakdown of the payload is shown in Appendix C.1.

A.2. enroll / reenroll

During the (re-)enroll exchange the EST-coaps client uses a CSR (Content-Format 286) request in the POST request payload. The Accept option tells the server that the client is expecting Content-Format 281 (PKCS#7) in the response. As shown in Appendix C.2, the CSR contains a ChallengePassword which is used for PoP linking (Section 4).

POST [2001:db8::2:321]:61616/est/sen
(Token: 0x45)
(Accept: 281)
(Content-Format: 286)

[The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text.]

3082018b30820131020100305c310b3009060355040613025553310b3009
06035504080c024341310b300906035504070c024c413114301206035504
0a0c0b6578616d706c6520496e63310c300a060355040b0c03496f54310f
300d060355040513065774313233343059301306072a8648ce3d02010608
2a8648ce3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f
028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75
f602f9152618f816a2b23b5638e59fd9a073303406092a864886f70d0109
0731270c2576437630292a264a4b4a3bc3a2c280c2992f3e3c2e2c3d6b6e
7634332323403d204e787e60303b06092a864886f70d01090e312e302c30
2a0603551d1104233021a01f06082b06010505070804a013301106092b06
010401b43b0a01040401020304300a06082a8648ce3d0403020348003045
02210092563a546463bd9ecff170d0fd1f2ef0d3d012160e5ee90cfffedab
ec9b9a38920220179f10a3436109051abad17590a09bc87c4dce5453a6fc
1135ale84eed754377

After verification of the CSR by the server, a 2.04 Changed response
with the issued certificate will be returned to the client.

2.04 Changed
(Token: 0x45)
(Content-Format: 281)

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
3082026e06092a864886f70d010702a082025f3082025b0201013100300b
06092a864886f70d010701a08202413082023d308201e2a0030201020208
7e7661d7b54e4632300a06082a8648ce3d040302305d310b300906035504
0613025553310b300906035504080c02434131143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3113301106035504030c0a3830322e3141522043413020170d31
39303133313131323931365a180f39393939313233313233353935395a30
5c310b3009060355040613025553310b300906035504080c024341310b30
0906035504070c024c4131143012060355040a0c0b6578616d706c652049
6e63310c300a060355040b0c03496f54310f300d06035504051306577431
3233343059301306072a8648ce3d020106082a8648ce3d03010703420004
c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351cc80c03f150bf50c
ff958d75419d81a6a245dffae790be95cf75f602f9152618f816a2b23b56
38e59fd9a3818a30818730090603551d1304023000301d0603551d0e0416
041496600d8716bf7fd0e752d0ac760777ad665d02a0301f0603551d2304
183016801468d16551f951bfc82a431d0d9f08bc2d205b1160300e060355
1d0f0101ff0404030205a0302a0603551d1104233021a01f06082b060105
05070804a013301106092b06010401b43b0a01040401020304300a06082a
8648ce3d0403020349003046022100c0d81996d2507d693f3c48eaa5ee94
91bda6db214099d98117c63b361374cd86022100a774989f4c321a5cf25d
832a4d336a08ad67df20f1506421188a0ade6d349236a1003100
```

The breakdown of the request and response is shown in Appendix C.2.

A.3. serverkeygen

In a serverkeygen exchange the CoAP POST request looks like

POST 192.0.2.1:8085/est/skg
(Token: 0xa5)
(Accept: 62)
(Content-Format: 286)

[The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text.]

3081d03078020100301631143012060355040a0c0b736b67206578616d70
6c653059301306072a8648ce3d020106082a8648ce3d03010703420004c8
b421f11c25e47e3ac57123bf2d9fdc494f028bc351cc80c03f150bf50cff
958d75419d81a6a245dffae790be95cf75f602f9152618f816a2b23b5638
e59fd9a000300a06082a8648ce3d040302034800304502207c553981b1fe
349249d8a3f50a0346336b7dfaa099cf74e1ec7a37a0a760485902210084
79295398774b2ff8e7e82abb0c17eaeef344a5088fa69fd63ee611850c34b
0a

The response would follow [I-D.ietf-core-multipart-ct] and could look
like

2.04 Changed
 (Token: 0xa5)
 (Content-Format: 62)

[The hexadecimal representations below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```

84                                     # array(4)
19 011C                             # unsigned(284)
58 8A                               # bytes(138)
308187020100301306072a8648ce3d020106082a8648ce3d030107046d30
6b020101042061336a86ac6e7af4a96f632830ad4e6aa0837679206094d7
679a01ca8c6f0c37a14403420004c8b421f11c25e47e3ac57123bf2d9fdc
494f028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95
cf75f602f9152618f816a2b23b5638e59fd9
19 0119                             # unsigned(281)
59 01D3                             # bytes(467)
308201cf06092a864886f70d010702a08201c0308201bc0201013100300b
06092a864886f70d010701a08201a23082019e30820144a0030201020209
00b3313e8f3fc9538e300a06082a8648ce3d040302301631143012060355
040a0c0b736b67206578616d706c65301e170d3139303930343037343430
335a170d3339303833303037343430335a301631143012060355040a0c0b
736b67206578616d706c653059301306072a8648ce3d020106082a8648ce
3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351
cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75f602f915
2618f816a2b23b5638e59fd9a37b307930090603551d1304023000302c06
096086480186f842010d041f161d4f70656e53534c2047656e6572617465
64204365727469666963617465301d0603551d0e0416041496600d8716bf
7fd0e752d0ac760777ad665d02a0301f0603551d2304183016801496600d
8716bf7fd0e752d0ac760777ad665d02a0300a06082a8648ce3d04030203
48003045022100e95bfa25a08976652246f2d96143da39fce0dc4c9b26b9
ccelf24164cc2b12b602201351fd8eea65764e3459d324e4345ff5b2a915
38c04976111796b3698bf6379ca1003100

```

The private key in the response above is without CMS EnvelopedData and has no additional encryption beyond DTLS (Section 5.8).

The breakdown of the request and response is shown in Appendix C.3

A.4. csrattrs

Below is a csrattrs exchange

REQ:
GET example.com:61616/est/att

RES:
2.05 Content
(Content-Format: 285)

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
307c06072b06010101011630220603883701311b131950617273652053455
420617320322e3939392e31206461746106092a864886f70d010907302c06
0388370231250603883703060388370413195061727365205345542061732
0322e3939392e32206461746106092b240303020801010b06096086480165
03040202
```

A 2.05 Content response should contain attributes which are relevant for the authenticated client. This example is copied from Section A.2 in [RFC7030], where the base64 representation is replaced with a hexadecimal representation of the equivalent binary format. The EST-coaps server returns attributes that the client can ignore if they are unknown to him.

Appendix B. EST-coaps Block message examples

Two examples are presented in this section:

1. a cacerts exchange shows the use of Block2 and the block headers
2. an enroll exchange shows the Block1 and Block2 size negotiation for request and response payloads.

The payloads are shown unencrypted. In practice the message contents would be binary formatted and transferred over an encrypted DTLS tunnel. The corresponding CoAP headers are only shown in Appendix B.1. Creating CoAP headers is assumed to be generally known.

B.1. cacerts

This section provides a detailed example of the messages using DTLS and BLOCK option Block2. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a cacerts exchange over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030] contains 639 bytes in binary in this example. The CoAP message adds

around 10 bytes in this example, the DTLS record around 29 bytes. To avoid IP fragmentation, the CoAP Block Option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 9 packets with a payload of 64 bytes each, followed by a last tenth packet of 63 bytes. The client sends an IPv6 packet containing a UDP datagram with DTLS record protection that encapsulates a CoAP request 10 times (one fragment of the request per block). The server returns an IPv6 packet containing a UDP datagram with the DTLS record that encapsulates the CoAP response. The CoAP request-response exchange with block option is shown below. Block Option is shown in a decomposed way (block-option:NUM/M/size) indicating the kind of Block Option (2 in this case) followed by a colon, and then the block number (NUM), the more bit (M = 0 in Block2 response means it is last block), and block size with exponent ($2^{*(SZX+4)}$) separated by slashes. The Length 64 is used with SZX=2. The CoAP Request is sent confirmable (CON) and the Content-Format of the response, even though not shown, is 281 (application/pkcs7-mime; smime-type=certs-only). The transfer of the 10 blocks with partially filled block NUM=9 is shown below

```

GET example.com:9085/est/crts (2:0/0/64) -->
      <-- (2:0/1/64) 2.05 Content
GET example.com:9085/est/crts (2:1/0/64) -->
      <-- (2:1/1/64) 2.05 Content
      |
      |
GET example.com:9085/est/crts (2:9/0/64) -->
      <-- (2:9/0/64) 2.05 Content

```

The header of the GET request looks like

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.1 GET)
Token = 0x9a      (client generated)
Options
  Option (Uri-Host)
    Option Delta = 0x3  (option# 3)
    Option Length = 0xB
    Option Value = "example.com"
  Option (Uri-Port)
    Option Delta = 0x4  (option# 3+4=7)
    Option Length = 0x2
    Option Value = 9085
  Option (Uri-Path)
    Option Delta = 0x4  (option# 7+4=11)
    Option Length = 0x3
    Option Value = "est"
  Option (Uri-Path)Uri-Path)
    Option Delta = 0x0  (option# 11+0=11)
    Option Length = 0x4
    Option Value = "crts"
  Option (Accept)
    Option Delta = 0x6  (option# 11+6=17)
    Option Length = 0x2
    Option Value = 281
Payload = [Empty]
```

The Uri-Host and Uri-Port Options can be omitted if they coincide with the transport protocol destination address and port respectively. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly.

For further detailing the CoAP headers, the first two and the last blocks are written out below. The header of the first Block2 response looks like

```
Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a      (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option# 12+11=23 Block2)
    Option Length = 0x1
    Option Value = 0x0A (block#=0, M=1, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
3082027b06092a864886f70d010702a082026c308202680201013100300b
06092a864886f70d010701a082024e3082024a308201f0a0030201020209
009189bc
```

The second Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a      (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option 12+11=23 Block2)
    Option Length = 0x1
    Option Value = 0x1A (block#=1, M=1, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
df9c99244b300a06082a8648ce3d0403023067310b300906035504061302
5553310b300906035504080c024341310b300906035504070c024c413114
30120603
```

The 10th and final Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x45      (2.05 Content)
Token = 0x9a     (copied from request by server)
Options
  Option
    Option Delta = 0xC (option# 12 Content-Format)
    Option Length = 0x2
    Option Value = 281
  Option
    Option Delta = 0xB (option# 12+11=23 Block2 )
    Option Length = 0x1
    Option Value = 0x92 (block#=9, M=0, SZX=2)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
Payload =
2ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f1353272f022047a28a
e5c7306163b3c3834bab3c103f743070594c089aaa0ac870cd13b902caa1
003100
```

B.2. enroll / reenroll

In this example, the requested Block2 size of 256 bytes, required by the client, is transferred to the server in the very first request message. The block size $256 = (2^{**}(\text{SZX}+4))$ which gives $\text{SZX}=4$. The notation for block numbering is the same as in Appendix B.1. The header fields and the payload are omitted for brevity.

```

POST [2001:db8::2:1]:61616/est/sen (CON) (1:0/1/256) {CSR (frag# 1)} -->
    <-- (ACK) (1:0/1/256) (2.31 Continue)
POST [2001:db8::2:1]:61616/est/sen (CON) (1:1/1/256) {CSR (frag# 2)} -->
    <-- (ACK) (1:1/1/256) (2.31 Continue)
    .
    .
    .
POST [2001:db8::2:1]:61616/est/sen (CON) (1:N1/0/256) {CSR(frag# N1+1)}-->
    |
    .....Immediate response .....
    |
    <-- (ACK) (1:N1/0/256) (2:0/1/256) (2.04 Changed) {Cert resp (frag# 1)}
POST [2001:db8::2:1]:61616/est/sen (CON) (2:1/0/256) -->
    <-- (ACK) (2:1/1/256) (2.04 Changed) {Cert resp (frag# 2)}
    .
    .
    .
POST [2001:db8::2:321]:61616/est/sen (CON) (2:N2/0/256) -->
    <-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp (frag# N2+1)}

```

Figure 5: EST-COAP enrollment with multiple blocks

N1+1 blocks have been transferred from client to the server and N2+1 blocks have been transferred from server to client.

Appendix C. Message content breakdown

This appendix presents the breakdown of the hexadecimal dumps of the binary payloads shown in Appendix A.

C.1. cacerts

The breakdown of cacerts response containing one root CA certificate is

Certificate:

```

Data:
  Version: 3 (0x2)
  Serial Number: 831953162763987486 (0xb8bb0fe604f6a1e)
  Signature Algorithm: ecdsa-with-SHA256
  Issuer: C=US, ST=CA, L=LA, O=Example Inc,
         OU=certification, CN=Root CA
  Validity
    Not Before: Jan 31 11:27:03 2019 GMT
    Not After : Jan 26 11:27:03 2039 GMT
  Subject: C=US, ST=CA, L=LA, O=Example Inc,
         OU=certification, CN=Root CA
  Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
    Public-Key: (256 bit)
    pub:
      04:0c:1b:1e:82:ba:8c:c7:26:80:97:3f:97:ed:b8:
      a0:c7:2a:b0:d4:05:f0:5d:4f:e2:9b:99:7a:14:cc:
      ce:89:00:83:13:d0:96:66:b6:ce:37:5c:59:5f:cc:
      8e:37:f8:e4:35:44:97:01:1b:e9:0e:56:79:4b:d9:
      1a:d9:51:ab:45
    ASN1 OID: prime256v1
    NIST CURVE: P-256
  X509v3 extensions:
    X509v3 Subject Key Identifier:
1D:F1:20:89:44:D7:7B:5F:1D:9D:CB:51:EE:24:4A:52:3F:3E:F5:DE
    X509v3 Authority Key Identifier:
      keyid:
1D:F1:20:89:44:D7:7B:5F:1D:9D:CB:51:EE:24:4A:52:3F:3E:F5:DE

    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Key Usage: critical
      Certificate Sign, CRL Sign
    X509v3 Subject Alternative Name:
      email:certify@example.com
  Signature Algorithm: ecdsa-with-SHA256
    30:45:02:20:2b:89:1d:d4:11:d0:7a:6d:6f:62:19:47:63:5b:
    a4:c4:31:65:29:6b:3f:63:37:26:f0:2e:51:ec:f4:64:bd:40:
    02:21:00:b4:be:8a:80:d0:86:75:f0:41:fb:c7:19:ac:f3:b3:
    9d:ed:c8:5d:c9:2b:30:35:86:8c:b2:da:a8:f0:5d:b1:96

```

C.2. enroll / reenroll

The breakdown of the enrollment request is

Certificate Request:

Data:

Version: 0 (0x0)

Subject: C=US, ST=CA, L=LA, O=example Inc,
OU=IoT/serialNumber=Wt1234

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

Attributes:

challengePassword: <256-bit PoP linking value>

Requested Extensions:

X509v3 Subject Alternative Name:

othername:<unsupported>

Signature Algorithm: ecdsa-with-SHA256

30:45:02:21:00:92:56:3a:54:64:63:bd:9e:cf:f1:70:d0:fd:
1f:2e:f0:d3:d0:12:16:0e:5e:e9:0c:ff:ed:ab:ec:9b:9a:38:
92:02:20:17:9f:10:a3:43:61:09:05:1a:ba:d1:75:90:a0:9b:
c8:7c:4d:ce:54:53:a6:fc:11:35:a1:e8:4e:ed:75:43:77

The CSR contains a ChallengePassword which is used for PoP linking (Section 4). The CSR also contains an id-on-hardwareModuleName hardware identifier to customize the returned certificate to the requesting device (See [RFC7299] and [I-D.moskowitz-ecdsa-pki]).

The breakdown of the issued certificate is

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 9112578475118446130 (0x7e7661d7b54e4632)

Signature Algorithm: ecdsa-with-SHA256

Issuer: C=US, ST=CA, O=Example Inc,
OU=certification, CN=802.1AR CA

Validity

Not Before: Jan 31 11:29:16 2019 GMT

Not After : Dec 31 23:59:59 9999 GMT

Subject: C=US, ST=CA, L=LA, O=example Inc,
OU=IoT/serialNumber=Wt1234

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

X509v3 Subject Key Identifier:

96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

X509v3 Authority Key Identifier:

keyid:

68:D1:65:51:F9:51:BF:C8:2A:43:1D:0D:9F:08:BC:2D:20:5B:11:60

X509v3 Key Usage: critical

Digital Signature, Key Encipherment

X509v3 Subject Alternative Name:

othername:<unsupported>

Signature Algorithm: ecdsa-with-SHA256

30:46:02:21:00:c0:d8:19:96:d2:50:7d:69:3f:3c:48:ea:a5:
ee:94:91:bd:a6:db:21:40:99:d9:81:17:c6:3b:36:13:74:cd:
86:02:21:00:a7:74:98:9f:4c:32:1a:5c:f2:5d:83:2a:4d:33:
6a:08:ad:67:df:20:f1:50:64:21:18:8a:0a:de:6d:34:92:36

C.3. serverkeygen

The following is the breakdown of the server-side key generation request.

Certificate Request:

Data:

Version: 0 (0x0)

Subject: O=skg example

Subject Public Key Info:

Public Key Algorithm: id-ecPublicKey

Public-Key: (256 bit)

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

Attributes:

a0:00

Signature Algorithm: ecdsa-with-SHA256

30:45:02:20:7c:55:39:81:b1:fe:34:92:49:d8:a3:f5:0a:03:

46:33:6b:7d:fa:a0:99:cf:74:e1:ec:7a:37:a0:a7:60:48:59:

02:21:00:84:79:29:53:98:77:4b:2f:f8:e7:e8:2a:bb:0c:17:

ea:ef:34:4a:50:88:fa:69:fd:63:ee:61:18:50:c3:4b:0a

Following is the breakdown of the private key content of the server-side key generation response.

Private-Key: (256 bit)

priv:

61:33:6a:86:ac:6e:7a:f4:a9:6f:63:28:30:ad:4e:

6a:a0:83:76:79:20:60:94:d7:67:9a:01:ca:8c:6f:

0c:37

pub:

04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:

9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:

0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:

be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:

56:38:e5:9f:d9

ASN1 OID: prime256v1

NIST CURVE: P-256

The following is the breakdown of the certificate in the server-side key generation response payload.

Certificate:

```
Data:
  Version: 3 (0x2)
  Serial Number:
    b3:31:3e:8f:3f:c9:53:8e
  Signature Algorithm: ecdsa-with-SHA256
  Issuer: O=skg example
  Validity
    Not Before: Sep  4 07:44:03 2019 GMT
    Not After : Aug 30 07:44:03 2039 GMT
  Subject: O=skg example
  Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
    Public-Key: (256 bit)
    pub:
      04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
      9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
      0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
      be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
      56:38:e5:9f:d9
    ASN1 OID: prime256v1
    NIST CURVE: P-256
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    Netscape Comment:
      OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0
    X509v3 Authority Key Identifier:
      keyid:
96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

  Signature Algorithm: ecdsa-with-SHA256
    30:45:02:21:00:e9:5b:fa:25:a0:89:76:65:22:46:f2:d9:61:
    43:da:39:fc:e0:dc:4c:9b:26:b9:cc:e1:f2:41:64:cc:2b:12:
    b6:02:20:13:51:fd:8e:ea:65:76:4e:34:59:d3:24:e4:34:5f:
    f5:b2:a9:15:38:c0:49:76:11:17:96:b3:69:8b:f6:37:9c
```

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: <http://www.sandelman.ca/>

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

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

S. Gerdes
O. Bergmann
C. Bormann
Universität Bremen TZI
G. Selander
Ericsson AB
L. Seitz
Combitech
4 June 2021

Datagram Transport Layer Security (DTLS) Profile for Authentication and
Authorization for Constrained Environments (ACE)
draft-ietf-ace-dtls-authorize-18

Abstract

This specification defines a profile of the ACE framework that allows constrained servers to delegate client authentication and authorization. The protocol relies on DTLS version 1.2 for communication security between entities in a constrained network using either raw public keys or pre-shared keys. A resource-constrained server can use this protocol to delegate management of authorization information to a trusted host with less severe limitations regarding processing power and memory.

Status of This Memo

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

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

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

This Internet-Draft will expire on 6 December 2021.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Protocol Overview	4
3. Protocol Flow	6
3.1. Communication Between the Client and the Authorization Server	6
3.2. Raw Public Key Mode	7
3.2.1. Access Token Retrieval from the Authorization Server	7
3.2.2. DTLS Channel Setup Between Client and Resource Server	9
3.3. PreSharedKey Mode	10
3.3.1. Access Token Retrieval from the Authorization Server	11
3.3.2. DTLS Channel Setup Between Client and Resource Server	15
3.4. Resource Access	17
4. Dynamic Update of Authorization Information	19
5. Token Expiration	20
6. Secure Communication with an Authorization Server	20
7. Security Considerations	21
7.1. Reuse of Existing Sessions	23
7.2. Multiple Access Tokens	23
7.3. Out-of-Band Configuration	23
8. Privacy Considerations	24
9. IANA Considerations	24
10. Acknowledgments	25
11. References	25
11.1. Normative References	25
11.2. Informative References	27
Authors' Addresses	28

1. Introduction

This specification defines a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use CoAP [RFC7252] over DTLS version 1.2 [RFC6347] to communicate. This specification uses DTLS 1.2 terminology, but later versions such as DTLS 1.3 can be used instead. The client obtains an access token, bound to a key (the proof-of-possession key), from an authorization server to prove its authorization to access protected resources hosted by the resource server. Also, the client and the resource server are provided by the authorization server with the necessary keying material to establish a DTLS session. The communication between client and authorization server may also be secured with DTLS. This specification supports DTLS with Raw Public Keys (RPK) [RFC7250] and with Pre-Shared Keys (PSK) [RFC4279]. How token introspection [RFC7662] is performed between RS and AS is out of scope for this specification.

The ACE framework requires that client and server mutually authenticate each other before any application data is exchanged. DTLS enables mutual authentication if both client and server prove their ability to use certain keying material in the DTLS handshake. The authorization server assists in this process on the server side by incorporating keying material (or information about keying material) into the access token, which is considered a "proof of possession" token.

In the RPK mode, the client proves that it can use the RPK bound to the token and the server shows that it can use a certain RPK.

The resource server needs access to the token in order to complete this exchange. For the RPK mode, the client must upload the access token to the resource server before initiating the handshake, as described in Section 5.10.1 of the ACE framework [I-D.ietf-ace-oauth-authz].

In the PSK mode, client and server show with the DTLS handshake that they can use the keying material that is bound to the access token. To transfer the access token from the client to the resource server, the "psk_identity" parameter in the DTLS PSK handshake may be used instead of uploading the token prior to the handshake.

As recommended in Section 5.8 of [I-D.ietf-ace-oauth-authz], this specification uses CBOR web tokens to convey claims within an access token issued by the server. While other formats could be used as well, those are out of scope for this document.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz] and in [I-D.ietf-ace-oauth-params].

The authorization information (authz-info) resource refers to the authorization information endpoint as specified in [I-D.ietf-ace-oauth-authz]. The term "claim" is used in this document with the same semantics as in [I-D.ietf-ace-oauth-authz], i.e., it denotes information carried in the access token or returned from introspection.

2. Protocol Overview

The CoAP-DTLS profile for ACE specifies the transfer of authentication information and, if necessary, authorization information between the client (C) and the resource server (RS) during setup of a DTLS session for CoAP messaging. It also specifies how the client can use CoAP over DTLS to retrieve an access token from the authorization server (AS) for a protected resource hosted on the resource server. As specified in Section 6.7 of [I-D.ietf-ace-oauth-authz], use of DTLS for one or both of these interactions is completely independent.

This profile requires the client to retrieve an access token for protected resource(s) it wants to access on the resource server as specified in [I-D.ietf-ace-oauth-authz]. Figure 1 shows the typical message flow in this scenario (messages in square brackets are optional):

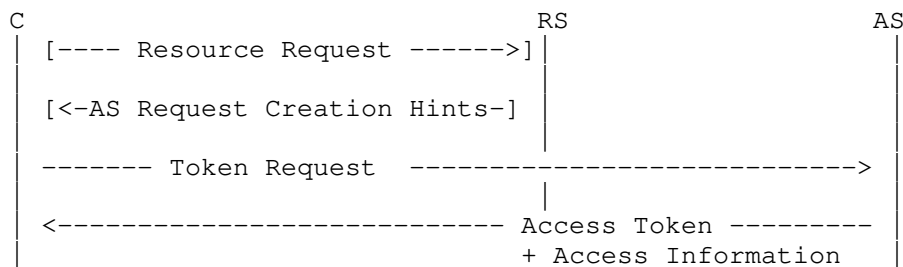


Figure 1: Retrieving an Access Token

To determine the authorization server in charge of a resource hosted at the resource server, the client can send an initial Unauthorized Resource Request message to the resource server. The resource server then denies the request and sends an AS Request Creation Hints message containing the address of its authorization server back to the client as specified in Section 5.3 of [I-D.ietf-ace-oauth-authz].

Once the client knows the authorization server's address, it can send an access token request to the token endpoint at the authorization server as specified in [I-D.ietf-ace-oauth-authz]. As the access token request as well as the response may contain confidential data, the communication between the client and the authorization server must be confidentiality-protected and ensure authenticity. The client is expected to have been registered at the authorization server as outlined in Section 4 of [I-D.ietf-ace-oauth-authz].

The access token returned by the authorization server can then be used by the client to establish a new DTLS session with the resource server. When the client intends to use an asymmetric proof-of-possession key in the DTLS handshake with the resource server, the client **MUST** upload the access token to the authz-info resource, i.e. the authz-info endpoint, on the resource server before starting the DTLS handshake, as described in Section 5.10.1 of [I-D.ietf-ace-oauth-authz]. In case the client uses a symmetric proof-of-possession key in the DTLS handshake, the procedure as above **MAY** be used, or alternatively, the access token **MAY** instead be transferred in the DTLS ClientKeyExchange message (see Section 3.3.2). In any case, DTLS **MUST** be used in a mode that provides replay protection.

Figure 2 depicts the common protocol flow for the DTLS profile after the client has retrieved the access token from the authorization server, AS.

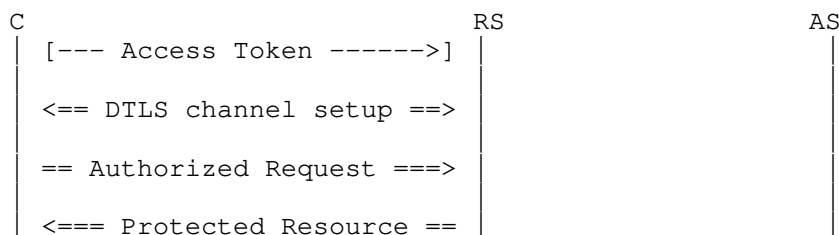


Figure 2: Protocol overview

3. Protocol Flow

The following sections specify how CoAP is used to interchange access-related data between the resource server, the client and the authorization server so that the authorization server can provide the client and the resource server with sufficient information to establish a secure channel, and convey authorization information specific for this communication relationship to the resource server.

Section 3.1 describes how the communication between the client (C) and the authorization server (AS) must be secured. Depending on the used CoAP security mode (see also Section 9 of [RFC7252], the Client-to-AS request, AS-to-Client response and DTLS session establishment carry slightly different information. Section 3.2 addresses the use of raw public keys while Section 3.3 defines how pre-shared keys are used in this profile.

3.1. Communication Between the Client and the Authorization Server

To retrieve an access token for the resource that the client wants to access, the client requests an access token from the authorization server. Before the client can request the access token, the client and the authorization server MUST establish a secure communication channel. This profile assumes that the keying material to secure this communication channel has securely been obtained either by manual configuration or in an automated provisioning process. The following requirements in alignment with Section 6.5 of [I-D.ietf-ace-oauth-authz] therefore must be met:

- * The client MUST securely have obtained keying material to communicate with the authorization server.
- * Furthermore, the client MUST verify that the authorization server is authorized to provide access tokens (including authorization information) about the resource server to the client, and that this authorization information about the authorization server is still valid.
- * Also, the authorization server MUST securely have obtained keying material for the client, and obtained authorization rules approved by the resource owner (RO) concerning the client and the resource server that relate to this keying material.

The client and the authorization server MUST use their respective keying material for all exchanged messages. How the security association between the client and the authorization server is bootstrapped is not part of this document. The client and the authorization server must ensure the confidentiality, integrity and authenticity of all exchanged messages within the ACE protocol.

Section 6 specifies how communication with the authorization server is secured.

3.2. Raw Public Key Mode

When the client uses raw public key authentication, the procedure is as described in the following.

3.2.1. Access Token Retrieval from the Authorization Server

After the client and the authorization server mutually authenticated each other and validated each other's authorization, the client sends a token request to the authorization server's token endpoint. The client MUST add a "req_cnf" object carrying either its raw public key or a unique identifier for a public key that it has previously made known to the authorization server. It is RECOMMENDED that the client uses DTLS with the same keying material to secure the communication with the authorization server, proving possession of the key as part of the token request. Other mechanisms for proving possession of the key may be defined in the future.

An example access token request from the client to the authorization server is depicted in Figure 3.

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  grant_type : client_credentials,
  audience   : "tempSensor4711",
  req_cnf    : {
    COSE_Key : {
      kty : EC2,
      crv : P-256,
      x   : h'e866c35f4c3c81bb96a1...',
      y   : h'2e25556be097c8778a20...'
    }
  }
}
```

Figure 3: Access Token Request Example for RPK Mode

The example shows an access token request for the resource identified by the string "tempSensor4711" on the authorization server using a raw public key.

The authorization server MUST check if the client that it communicates with is associated with the RPK in the "req_cnf" parameter before issuing an access token to it. If the authorization server determines that the request is to be authorized according to the respective authorization rules, it generates an access token response for the client. The access token MUST be bound to the RPK of the client by means of the "cnf" claim.

The response MUST contain an "ace_profile" parameter if the "ace_profile" parameter in the request is empty, and MAY contain this parameter otherwise (see Section 5.8.2 of [I-D.ietf-ace-oauth-authz]). This parameter is set to "coap_dtls" to indicate that this profile MUST be used for communication between the client and the resource server. The response also contains an access token with information for the resource server about the client's public key. The authorization server MUST return in its response the parameter "rs_cnf" unless it is certain that the client already knows the public key of the resource server. The authorization server MUST ascertain that the RPK specified in "rs_cnf" belongs to the resource server that the client wants to communicate with. The authorization server MUST protect the integrity of the access token such that the resource server can detect unauthorized changes. If the access token contains confidential data, the authorization server MUST also protect the confidentiality of the access token.

The client MUST ascertain that the access token response belongs to a certain previously sent access token request, as the request may specify the resource server with which the client wants to communicate.

An example access token response from the authorization server to the client is depicted in Figure 4. Here, the contents of the "access_token" claim have been truncated to improve readability. The response comprises access information for the client that contains the server's public key in the "rs_cnf" parameter. Caching proxies process the Max-Age option in the CoAP response which has a default value of 60 seconds (Section 5.6.1 of [RFC7252]). The authorization server SHOULD adjust the Max-Age option such that it does not exceed the "expires_in" parameter to avoid stale responses.

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 3560
Payload:
{
  access_token : b64'SlAV32hkKG...
    (remainder of CWT omitted for brevity;
    CWT contains the client's RPK in the cnf claim)',
  expires_in : 3600,
  rs_cnf      : {
    COSE_Key : {
      kty : EC2,
      crv : P-256,
      x   : h'd7cc072de2205bdc1537...',
      y   : h'f95e1d4b851a2cc80fff...'
    }
  }
}
```

Figure 4: Access Token Response Example for RPK Mode

3.2.2. DTLS Channel Setup Between Client and Resource Server

Before the client initiates the DTLS handshake with the resource server, the client MUST send a "POST" request containing the obtained access token to the authz-info resource hosted by the resource server. After the client receives a confirmation that the resource server has accepted the access token, it proceeds to establish a new DTLS channel with the resource server. The client MUST use its correct public key in the DTLS handshake. If the authorization server has specified a "cnf" field in the access token response, the client MUST use this key. Otherwise, the client MUST use the public key that it specified in the "req_cnf" of the access token request. The client MUST specify this public key in the SubjectPublicKeyInfo structure of the DTLS handshake as described in [RFC7250].

If the client does not have the keying material belonging to the public key, the client MAY try to send an access token request to the AS where it specifies its public key in the "req_cnf" parameter. If the AS still specifies a public key in the response that the client does not have, the client SHOULD re-register with the authorization server to establish a new client public key. This process is out of scope for this document.

To be consistent with [RFC7252], which allows for shortened MAC tags in constrained environments, an implementation that supports the RPK mode of this profile MUST at least support the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. As discussed in

[RFC7748], new ECC curves have been defined recently that are considered superior to the so-called NIST curves. Implementations of this profile therefore MUST implement support for curve25519 (cf. [RFC8032], [RFC8422]) as this curve said to be efficient and less dangerous regarding implementation errors than the secp256r1 curve mandated in [RFC7252].

The resource server MUST check if the access token is still valid, if the resource server is the intended destination (i.e., the audience) of the token, and if the token was issued by an authorized authorization server (see also section 5.10.1.1 of [I-D.ietf-ace-oauth-authz]). The access token is constructed by the authorization server such that the resource server can associate the access token with the Client's public key. The "cnf" claim MUST contain either the client's RPK or, if the key is already known by the resource server (e.g., from previous communication), a reference to this key. If the authorization server has no certain knowledge that the Client's key is already known to the resource server, the Client's public key MUST be included in the access token's "cnf" parameter. If CBOR web tokens [RFC8392] are used (as recommended in [I-D.ietf-ace-oauth-authz]), keys MUST be encoded as specified in [RFC8747]. A resource server MUST have the capacity to store one access token for every proof-of-possession key of every authorized client.

The raw public key used in the DTLS handshake with the client MUST belong to the resource server. If the resource server has several raw public keys, it needs to determine which key to use. The authorization server can help with this decision by including a "cnf" parameter in the access token that is associated with this communication. In this case, the resource server MUST use the information from the "cnf" field to select the proper keying material.

Thus, the handshake only finishes if the client and the resource server are able to use their respective keying material.

3.3. PreSharedKey Mode

When the client uses pre-shared key authentication, the procedure is as described in the following.

3.3.1. Access Token Retrieval from the Authorization Server

To retrieve an access token for the resource that the client wants to access, the client MAY include a "cnf" object carrying an identifier for a symmetric key in its access token request to the authorization server. This identifier can be used by the authorization server to determine the shared secret to construct the proof-of-possession token. The authorization server MUST check if the identifier refers to a symmetric key that was previously generated by the authorization server as a shared secret for the communication between this client and the resource server. If no such symmetric key was found, the authorization server MUST generate a new symmetric key that is returned in its response to the client.

The authorization server MUST determine the authorization rules for the client it communicates with as defined by the resource owner and generate the access token accordingly. If the authorization server authorizes the client, it returns an AS-to-Client response. If the "ace_profile" parameter is present, it is set to "coap_dtls". The authorization server MUST ascertain that the access token is generated for the resource server that the client wants to communicate with. Also, the authorization server MUST protect the integrity of the access token to ensure that the resource server can detect unauthorized changes. If the token contains confidential data such as the symmetric key, the confidentiality of the token MUST also be protected. Depending on the requested token type and algorithm in the access token request, the authorization server adds access information to the response that provides the client with sufficient information to setup a DTLS channel with the resource server. The authorization server adds a "cnf" parameter to the access information carrying a "COSE_Key" object that informs the client about the shared secret that is to be used between the client and the resource server. To convey the same secret to the resource server, the authorization server can include it directly in the access token by means of the "cnf" claim or provide sufficient information to enable the resource server to derive the shared secret from the access token. As an alternative, the resource server MAY use token introspection to retrieve the keying material for this access token directly from the authorization server.

An example access token request for an access token with a symmetric proof-of-possession key is illustrated in Figure 5.

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  audience      : "smokeSensor1807",
}
```

Figure 5: Example Access Token Request, (implicit) symmetric PoP-key

A corresponding example access token response is illustrated in Figure 6. In this example, the authorization server returns a 2.01 response containing a new access token (truncated to improve readability) and information for the client, including the symmetric key in the cnf claim. The information is transferred as a CBOR data structure as specified in [I-D.ietf-ace-oauth-authz].

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 85800
Payload:
{
  access_token : h'd08343a10...
  (remainder of CWT omitted for brevity)
  token_type   : PoP,
  expires_in   : 86400,
  profile      : coap_dtls,
  cnf          : {
    COSE_Key : {
      kty : symmetric,
      kid : h'3d027833fc6267ce',
      k   : h'73657373696f6e6b6579'
    }
  }
}
```

Figure 6: Example Access Token Response, symmetric PoP-key

The access token also comprises a "cnf" claim. This claim usually contains a "COSE_Key" object [RFC8152] that carries either the symmetric key itself or a key identifier that can be used by the resource server to determine the secret key it shares with the client. If the access token carries a symmetric key, the access token MUST be encrypted using a "COSE_Encrypt0" structure (see section 7.1 of [RFC8392]). The authorization server MUST use the keying material shared with the resource server to encrypt the token.

The "cnf" structure in the access token is provided in Figure 7.

```
cnf : {  
  COSE_Key : {  
    kty : symmetric,  
    kid : h'3d027833fc6267ce'  
  }  
}
```

Figure 7: Access Token without Keying Material

A response that declines any operation on the requested resource is constructed according to Section 5.2 of [RFC6749], (cf. Section 5.8.3. of [I-D.ietf-ace-oauth-authz]). Figure 8 shows an example for a request that has been rejected due to invalid request parameters.

```
4.00 Bad Request  
Content-Format: application/ace+cbor  
Payload:  
{  
  error : invalid_request  
}
```

Figure 8: Example Access Token Response With Reject

The method for how the resource server determines the symmetric key from an access token containing only a key identifier is application-specific; the remainder of this section provides one example.

The authorization server and the resource server are assumed to share a key derivation key used to derive the symmetric key shared with the client from the key identifier in the access token. The key derivation key may be derived from some other secret key shared between the authorization server and the resource server. This key needs to be securely stored and processed in the same way as the key used to protect the communication between the authorization server and the resource server.

Knowledge of the symmetric key shared with the client must not reveal any information about the key derivation key or other secret keys shared between the authorization server and resource server.

In order to generate a new symmetric key to be used by client and resource server, the authorization server generates a new key identifier which **MUST** be unique among all key identifiers used by the authorization server for this resource server. The authorization server then uses the key derivation key shared with the resource server to derive the symmetric key as specified below. Instead of providing the keying material in the access token, the authorization

server includes the key identifier in the "kid" parameter, see Figure 7. This key identifier enables the resource server to calculate the symmetric key used for the communication with the client using the key derivation key and a KDF to be defined by the application, for example HKDF-SHA-256. The key identifier picked by the authorization server MUST be unique for each access token where a unique symmetric key is required.

In this example, HKDF consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]. The symmetric key is derived from the key identifier, the key derivation key and other data:

```
OKM = HKDF(salt, IKM, info, L),
```

where:

- * OKM, the output keying material, is the derived symmetric key
- * salt is the empty byte string
- * IKM, the input keying material, is the key derivation key as defined above
- * info is the serialization of a CBOR array consisting of ([RFC8610]):

```
info = [  
    type : tstr,  
    L : uint,  
    access_token: bytes  
]
```

where:

- * type is set to the constant text string "ACE-CoAP-DTLS-key-derivation",
- * L is the size of the symmetric key in bytes,
- * access_token is the content of the "access_token" field as transferred from the authorization server to the resource server.

All CBOR data types are encoded in CBOR using preferred serialization and deterministic encoding as specified in Section 4 of [RFC8949]. This implies in particular that the "type" and "L" components use the minimum length encoding. The content of the "access_token" field is treated as opaque data for the purpose of key derivation.

Use of a unique (per resource server) "kid" and the use of a key derivation IKM that MUST be unique per authorization server/resource server pair as specified above will ensure that the derived key is not shared across multiple clients. However, to provide variation in the derived key across different tokens used by the same client, it is additionally RECOMMENDED to include the "iat" claim and either the "exp" or "exp" claims in the access token.

3.3.2. DTLS Channel Setup Between Client and Resource Server

When a client receives an access token response from an authorization server, the client MUST check if the access token response is bound to a certain previously sent access token request, as the request may specify the resource server with which the client wants to communicate.

The client checks if the payload of the access token response contains an "access_token" parameter and a "cnf" parameter. With this information the client can initiate the establishment of a new DTLS channel with a resource server. To use DTLS with pre-shared keys, the client follows the PSK key exchange algorithm specified in Section 2 of [RFC4279] using the key conveyed in the "cnf" parameter of the AS response as PSK when constructing the premaster secret. To be consistent with the recommendations in [RFC7252], a client in the PSK mode MUST support the cipher suite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655].

In PreSharedKey mode, the knowledge of the shared secret by the client and the resource server is used for mutual authentication between both peers. Therefore, the resource server must be able to determine the shared secret from the access token. Following the general ACE authorization framework, the client can upload the access token to the resource server's authz-info resource before starting the DTLS handshake. The client then needs to indicate during the DTLS handshake which previously uploaded access token it intends to use. To do so, it MUST create a "COSE_Key" structure with the "kid" that was conveyed in the "rs_cnf" claim in the token response from the authorization server and the key type "symmetric". This structure then is included as the only element in the "cnf" structure whose CBOR serialization is used as value for "psk_identity" as shown in Figure 9.

```
{ cnf : {  
  COSE_Key : {  
    kty: symmetric,  
    kid : h'3d027833fc6267ce'  
  }  
}
```

Figure 9: Access token containing a single kid parameter

The actual CBOR serialization for the data structure from Figure 9 as sequence of bytes in hexadecimal notation will be:

A1 08 A1 01 A2 01 04 02 48 3D 02 78 33 FC 62 67 CE

As an alternative to the access token upload, the client can provide the most recent access token in the "psk_identity" field of the ClientKeyExchange message. To do so, the client MUST treat the contents of the "access_token" field from the AS-to-Client response as opaque data as specified in Section 4.2 of [RFC7925] and not perform any re-coding. This allows the resource server to retrieve the shared secret directly from the "cnf" claim of the access token.

If a resource server receives a ClientKeyExchange message that contains a "psk_identity" with a length greater than zero, it MUST parse the contents of the "psk_identity" field as CBOR data structure and process the contents as following:

- * If the data contains a "cnf" field with a "COSE_Key" structure with a "kid", the resource server continues the DTLS handshake with the associated key that corresponds to this kid.
- * If the data comprises additional CWT information, this information must be stored as an access token for this DTLS association before continuing with the DTLS handshake.

If the contents of the "psk_identity" do not yield sufficient information to select a valid access token for the requesting client, the resource server aborts the DTLS handshake with an "illegal_parameter" alert.

When the resource server receives an access token, it MUST check if the access token is still valid, if the resource server is the intended destination (i.e., the audience of the token), and if the token was issued by an authorized authorization server. This specification implements access tokens as proof-of-possession tokens. Therefore, the access token is bound to a symmetric PoP key that is used as shared secret between the client and the resource server. A

resource server MUST have the capacity to store one access token for every proof-of-possession key of every authorized client. The resource server may use token introspection [RFC7662] on the access token to retrieve more information about the specific token. The use of introspection is out of scope for this specification.

While the client can retrieve the shared secret from the contents of the "cnf" parameter in the AS-to-Client response, the resource server uses the information contained in the "cnf" claim of the access token to determine the actual secret when no explicit "kid" was provided in the "psk_identity" field. If key derivation is used, the "cnf" claim MUST contain a "kid" parameter to be used by the server as the IKM for key derivation as described above.

3.4. Resource Access

Once a DTLS channel has been established as described in Section 3.2 or Section 3.3, respectively, the client is authorized to access resources covered by the access token it has uploaded to the authz-info resource hosted by the resource server.

With the successful establishment of the DTLS channel, the client and the resource server have proven that they can use their respective keying material. An access token that is bound to the client's keying material is associated with the channel. According to Section 5.10.1 of [I-D.ietf-ace-oauth-authz], there should be only one access token for each client. New access tokens issued by the authorization server SHOULD replace previously issued access tokens for the respective client. The resource server therefore needs a common understanding with the authorization server how access tokens are ordered. The authorization server may, e.g., specify a "cti" claim for the access token (see Section 5.9.4 of [I-D.ietf-ace-oauth-authz]) to employ a strict order.

Any request that the resource server receives on a DTLS channel that is tied to an access token via its keying material MUST be checked against the authorization rules that can be determined with the access token. The resource server MUST check for every request if the access token is still valid. If the token has expired, the resource server MUST remove it. Incoming CoAP requests that are not authorized with respect to any access token that is associated with the client MUST be rejected by the resource server with 4.01 response. The response SHOULD include AS Request Creation Hints as described in Section 5.2 of [I-D.ietf-ace-oauth-authz].

The resource server MUST NOT accept an incoming CoAP request as authorized if any of the following fails:

1. The message was received on a secure channel that has been established using the procedure defined in this document.
2. The authorization information tied to the sending client is valid.
3. The request is destined for the resource server.
4. The resource URI specified in the request is covered by the authorization information.
5. The request method is an authorized action on the resource with respect to the authorization information.

Incoming CoAP requests received on a secure DTLS channel that are not thus authorized **MUST** be rejected according to Section 5.10.1.1 of [I-D.ietf-ace-oauth-authz]

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and
2. with response code 4.05 (Method Not Allowed) when the resource URI specified in the request covered by the authorization information but not the requested action.

The client **MUST** ascertain that its keying material is still valid before sending a request or processing a response. If the client recently has updated the access token (see Section 4), it must be prepared that its request is still handled according to the previous authorization rules as there is no strict ordering between access token uploads and resource access messages. See also Section 7.2 for a discussion of access token processing.

If the client gets an error response containing AS Request Creation Hints (cf. Section 5.3 of [I-D.ietf-ace-oauth-authz]) as response to its requests, it **SHOULD** request a new access token from the authorization server in order to continue communication with the resource server.

Unauthorized requests that have been received over a DTLS session **SHOULD** be treated as non-fatal by the resource server, i.e., the DTLS session **SHOULD** be kept alive until the associated access token has expired.

4. Dynamic Update of Authorization Information

Resource servers must only use a new access token to update the authorization information for a DTLS session if the keying material that is bound to the token is the same that was used in the DTLS handshake. By associating the access tokens with the identifier of an existing DTLS session, the authorization information can be updated without changing the cryptographic keys for the DTLS communication between the client and the resource server, i.e. an existing session can be used with updated permissions.

The client can therefore update the authorization information stored at the resource server at any time without changing an established DTLS session. To do so, the client requests a new access token from the authorization server for the intended action on the respective resource and uploads this access token to the authz-info resource on the resource server.

Figure 10 depicts the message flow where the client requests a new access token after a security association between the client and the resource server has been established using this protocol. If the client wants to update the authorization information, the token request MUST specify the key identifier of the proof-of-possession key used for the existing DTLS channel between the client and the resource server in the "kid" parameter of the Client-to-AS request. The authorization server MUST verify that the specified "kid" denotes a valid verifier for a proof-of-possession token that has previously been issued to the requesting client. Otherwise, the Client-to-AS request MUST be declined with the error code "unsupported_pop_key" as defined in Section 5.8.3 of [I-D.ietf-ace-oauth-authz].

When the authorization server issues a new access token to update existing authorization information, it MUST include the specified "kid" parameter in this access token. A resource server MUST replace the authorization information of any existing DTLS session that is identified by this key identifier with the updated authorization information.

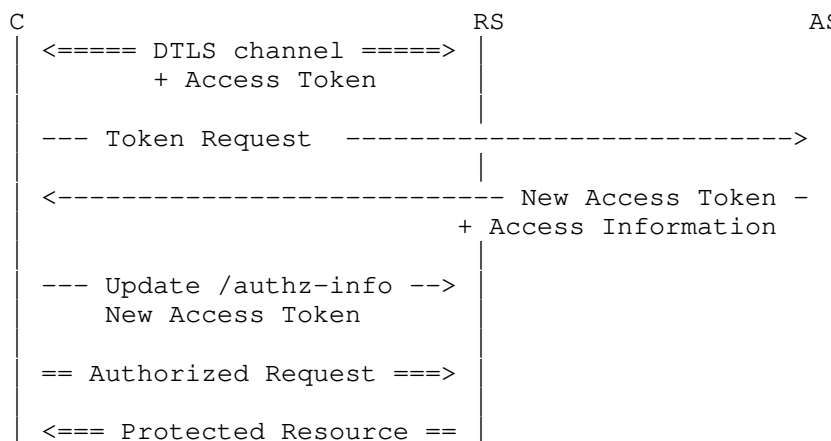


Figure 10: Overview of Dynamic Update Operation

5. Token Expiration

The resource server MUST delete access tokens that are no longer valid. DTLS associations that have been setup in accordance with this profile are always tied to specific tokens (which may be exchanged with a dynamic update as described in Section 4). As tokens may become invalid at any time (e.g., because they have expired), the association may become useless at some point. A resource server therefore MUST terminate existing DTLS association after the last access token associated with this association has expired.

As specified in Section 5.10.3 of [I-D.ietf-ace-oauth-authz], the resource server MUST notify the client with an error response with code 4.01 (Unauthorized) for any long running request before terminating the association.

6. Secure Communication with an Authorization Server

As specified in the ACE framework (Sections 5.8 and 5.9 of [I-D.ietf-ace-oauth-authz]), the requesting entity (the resource server and/or the client) and the authorization server communicate via the token endpoint or introspection endpoint. The use of CoAP and DTLS for this communication is RECOMMENDED in this profile. Other protocols fulfilling the security requirements defined in Section 5 of [I-D.ietf-ace-oauth-authz] MAY be used instead.

How credentials (e.g., PSK, RPK, X.509 cert) for using DTLS with the authorization server are established is out of scope for this profile.

If other means of securing the communication with the authorization server are used, the communication security requirements from Section 6.2 of [I-D.ietf-ace-oauth-authz] remain applicable.

7. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. As it follows this framework's general approach, the general security considerations from Section 6 of [I-D.ietf-ace-oauth-authz] also apply to this profile.

The authorization server must ascertain that the keying material for the client that it provides to the resource server actually is associated with this client. Malicious clients may hand over access tokens containing their own access permissions to other entities. This problem cannot be completely eliminated. Nevertheless, in RPK mode it should not be possible for clients to request access tokens for arbitrary public keys: if the client can cause the authorization server to issue a token for a public key without proving possession of the corresponding private key, this allows for identity misbinding attacks where the issued token is usable by an entity other than the intended one. The authorization server therefore at some point needs to validate that the client can actually use the private key corresponding to the client's public key.

When using pre-shared keys provisioned by the authorization server, the security level depends on the randomness of PSK, and the security of the TLS cipher suite and key exchange algorithm. As this specification targets at constrained environments, message payloads exchanged between the client and the resource server are expected to be small and rare. CoAP [RFC7252] mandates the implementation of cipher suites with abbreviated, 8-byte tags for message integrity protection. For consistency, this profile requires implementation of the same cipher suites. For application scenarios where the cost of full-width authentication tags is low compared to the overall amount of data being transmitted, the use of cipher suites with 16-byte integrity protection tags is preferred.

The PSK mode of this profile offers a distribution mechanism to convey authorization tokens together with a shared secret to a client and a server. As this specification aims at constrained devices and uses CoAP [RFC7252] as transfer protocol, at least the cipher suite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655] should be supported. The access tokens and the corresponding shared secrets generated by the authorization server are expected to be sufficiently short-lived to provide similar forward-secrecy properties to using ephemeral Diffie-Hellman (DHE) key exchange mechanisms. For longer-lived access tokens, DHE cipher suites should be used, i.e., cipher suites of the form TLS_DHE_PSK_*.

Constrained devices that use DTLS [RFC6347] are inherently vulnerable to Denial of Service (DoS) attacks as the handshake protocol requires creation of internal state within the device. This is specifically of concern where an adversary is able to intercept the initial cookie exchange and interject forged messages with a valid cookie to continue with the handshake. A similar issue exists with the unprotected authorization information endpoint when the resource server needs to keep valid access tokens for a long time. Adversaries could fill up the constrained resource server's internal storage for a very long time with interjected or otherwise retrieved valid access tokens. To mitigate against this, the resource server should set a time boundary until an access token that has not been used until then will be deleted.

The protection of access tokens that are stored in the authorization information endpoint depends on the keying material that is used between the authorization server and the resource server: The resource server must ensure that it processes only access tokens that are (encrypted and) integrity-protected by an authorization server that is authorized to provide access tokens for the resource server.

7.1. Reuse of Existing Sessions

To avoid the overhead of a repeated DTLS handshake, [RFC7925] recommends session resumption [RFC8446] to reuse session state from an earlier DTLS association and thus requires client side implementation. In this specification, the DTLS session is subject to the authorization rules denoted by the access token that was used for the initial setup of the DTLS association. Enabling session resumption would require the server to transfer the authorization information with the session state in an encrypted SessionTicket to the client. Assuming that the server uses long-lived keying material, this could open up attacks due to the lack of forward secrecy. Moreover, using this mechanism, a client can resume a DTLS session without proving the possession of the PoP key again. Therefore, session resumption should be used only in combination with reasonably short-lived PoP keys.

Since renegotiation of DTLS associations is prone to attacks as well, [RFC7925] requires clients to decline any renegotiation attempt. A server that wants to initiate re-keying therefore SHOULD periodically force a full handshake.

7.2. Multiple Access Tokens

Developers SHOULD avoid using multiple access tokens for a client (see also section 5.10.1 of [I-D.ietf-ace-oauth-authz]).

Even when a single access token per client is used, an attacker could compromise the dynamic update mechanism for existing DTLS connections by delaying or reordering packets destined for the authz-info endpoint. Thus, the order in which operations occur at the resource server (and thus which authorization info is used to process a given client request) cannot be guaranteed. Especially in the presence of later-issued access tokens that reduce the client's permissions from the initial access token, it is impossible to guarantee that the reduction in authorization will take effect prior to the expiration of the original token.

7.3. Out-of-Band Configuration

To communicate securely, the authorization server, the client and the resource server require certain information that must be exchanged outside the protocol flow described in this document. The authorization server must have obtained authorization information concerning the client and the resource server that is approved by the resource owner as well as corresponding keying material. The resource server must have received authorization information approved by the resource owner concerning its authorization managers and the

respective keying material. The client must have obtained authorization information concerning the authorization server approved by its owner as well as the corresponding keying material. Also, the client's owner must have approved of the client's communication with the resource server. The client and the authorization server must have obtained a common understanding how this resource server is identified to ensure that the client obtains access token and keying material for the correct resource server. If the client is provided with a raw public key for the resource server, it must be ascertained to which resource server (which identifier and authorization information) the key is associated. All authorization information and keying material must be kept up to date.

8. Privacy Considerations

This privacy considerations from Section 7 of the [I-D.ietf-ace-oauth-authz] apply also to this profile.

An unprotected response to an unauthorized request may disclose information about the resource server and/or its existing relationship with the client. It is advisable to include as little information as possible in an unencrypted response. When a DTLS session between an authenticated client and the resource server already exists, more detailed information MAY be included with an error response to provide the client with sufficient information to react on that particular error.

Also, unprotected requests to the resource server may reveal information about the client, e.g., which resources the client attempts to request or the data that the client wants to provide to the resource server. The client SHOULD NOT send confidential data in an unprotected request.

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

9. IANA Considerations

The following registrations are done for the ACE OAuth Profile Registry following the procedure specified in [I-D.ietf-ace-oauth-authz].

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

Profile name: coap_dtls

Profile Description: Profile for delegating client authentication and authorization in a constrained environment by establishing a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes.

Profile ID: TBD (suggested: 1)

Change Controller: IESG

Reference: [RFC-XXXX]

10. Acknowledgments

Special thanks to Jim Schaad for his contributions and reviews of this document and to Ben Kaduk for his thorough reviews of this document. Thanks also to Paul Kyzivat for his review. The authors also would like to thank Marco Tiloca for his contributions.

Ludwig Seitz worked on this document as part of the CelticNext projects CyberWI, and CRITISEC with funding from Vinnova.

11. References

11.1. Normative References

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-41, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-41.txt>>.

[I-D.ietf-ace-oauth-params]

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-15, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-15.txt>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <<https://www.rfc-editor.org/info/rfc4279>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [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>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <<https://www.rfc-editor.org/info/rfc7251>>.
- [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>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

- [RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

11.2. Informative References

- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <<https://www.rfc-editor.org/info/rfc6655>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

[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>>.

Authors' Addresses

Stefanie Gerdes
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

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

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

Göran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Ludwig Seitz
Combitech
Djäknegatan 31
SE-211 35 Malmö
Sweden

Email: ludwig.seitz@combitech.com

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

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

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

Abstract

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

Discussion Venues

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 26 June 2022.

Copyright Notice

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

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

Table of Contents

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

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

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

1. Introduction

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

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

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

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

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

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

1.1. Terminology

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

Readers are expected to be familiar with:

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

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

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

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

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

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

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

This document additionally uses the following terminology:

- * Transport profile, to indicate a profile of ACE as per Section 5.8.4.3 of [I-D.ietf-ace-oauth-authz]. A transport profile specifies the communication protocol and communication security protocol between an ACE Client and Resource Server, as well as proof-of-possession methods, if it supports proof-of-possession access tokens, etc. Transport profiles of ACE include, for instance, [I-D.ietf-ace-oscore-profile], [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-mqtt-tls-profile].
- * Application profile, that defines how applications enforce and use supporting security services they require. These services may include, for instance, provisioning, revocation and distribution of keying material. An application profile may define specific procedures and message formats.

2. Overview

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

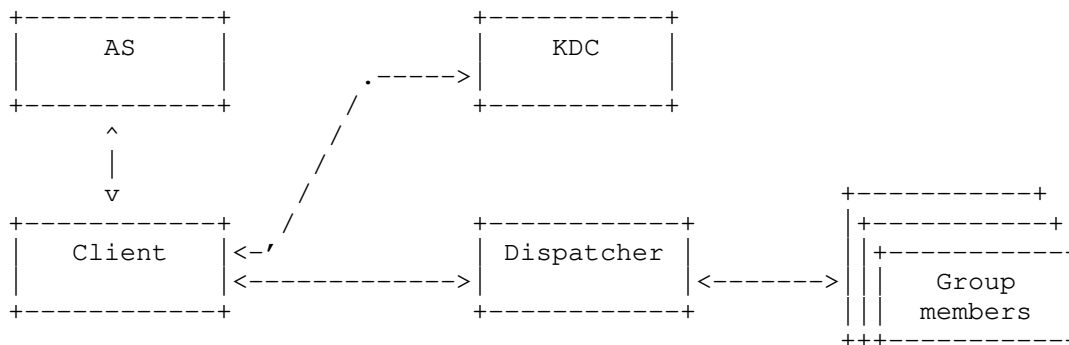


Figure 1: Key Distribution Participants

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

- * Client (C): node that wants to join a group and take part in group communication with other group members. Within the group, the Client can have different roles.
- * Authorization Server (AS): as per the AS defined in the ACE Framework, it enforces access policies, and knows if a node is allowed to join a given group with write and/or read rights.

- * Key Distribution Center (KDC): maintains the keying material to protect group communications, and provides it to Clients authorized to join a given group. During the first part of the exchange (Section 3), it takes the role of the RS in the ACE Framework. During the second part (Section 4), which is not based on the ACE Framework, it distributes the keying material. In addition, it provides the latest keying material to group members when requested or, if required by the application, when membership changes.
- * Dispatcher: entity through which the Clients communicate with the group, when sending a message intended to multiple group members. That is, the Dispatcher distributes such a one-to-many message to the group members as intended recipients. A single-recipient message intended to only one group member may be delivered by alternative means, with no assistance from the Dispatcher.

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

This document specifies a mechanism for:

- * Authorizing a Client to join the group (Section 3), and providing it with the group keying material to communicate with the other group members (Section 4).
- * Allowing a group member to retrieve group keying material (Section 4.8.1.1 and Section 4.8.2.1).
- * Allowing a group member to retrieve public keys of other group members (Section 4.4.1.1) and to provide an updated public key (Section 4.9.1.1).
- * Allowing a group member to leave the group (Section 5).
- * Evicting a group member from the group (Section 5).
- * Renewing and re-distributing the group keying material (rekeying) upon a membership change in the group (Section 4.8.3.1 and Section 5).

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

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

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

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

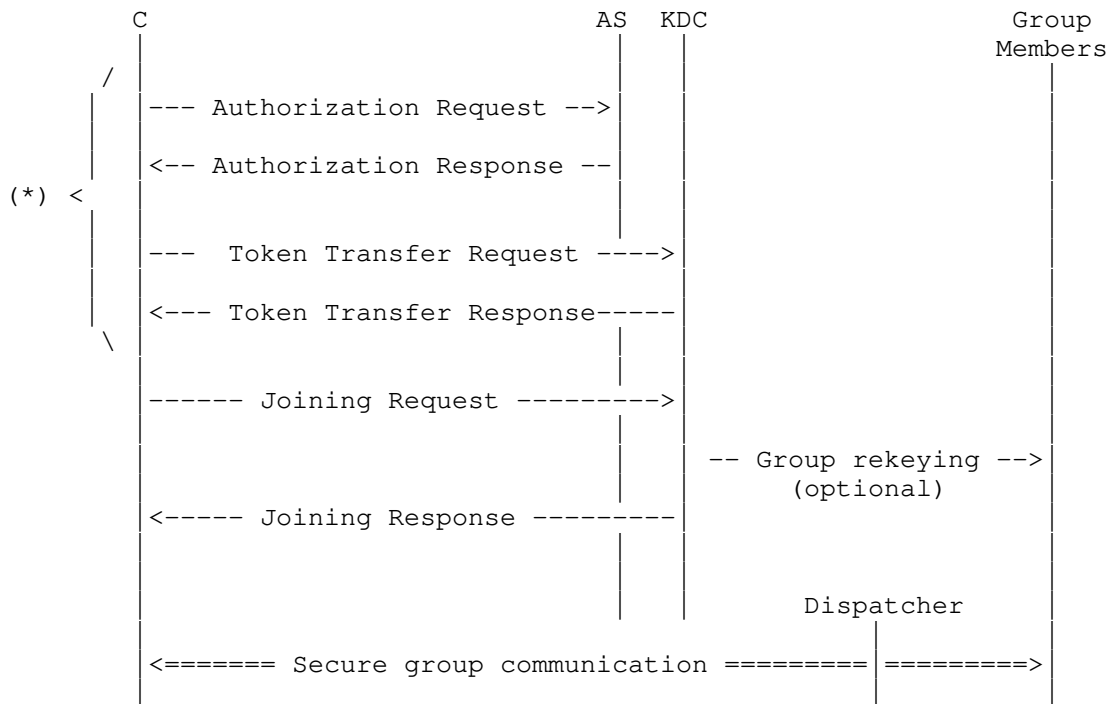
Once this exchange is completed, the joining node MUST have a secure communication association established with the KDC, before joining a group under that KDC.

This exchange and the following secure communications between the Client and the KDC MUST occur in accordance with the transport profile of ACE used between Client and KDC, such as the DTLS transport profile [I-D.ietf-ace-dtls-authorize] and OSCORE transport profile [I-D.ietf-ace-oscore-profile] of ACE.

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

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

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



(*) Defined in the ACE framework

Figure 2: Message Flow Upon New Node's Joining

3. Authorization to Join a Group

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

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

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

The transport profile of ACE also defines a number of details such as the communication and security protocols used with the KDC (see Appendix C of [I-D.ietf-ace-oauth-authz]).

Figure 3 gives an overview of the exchange described above.

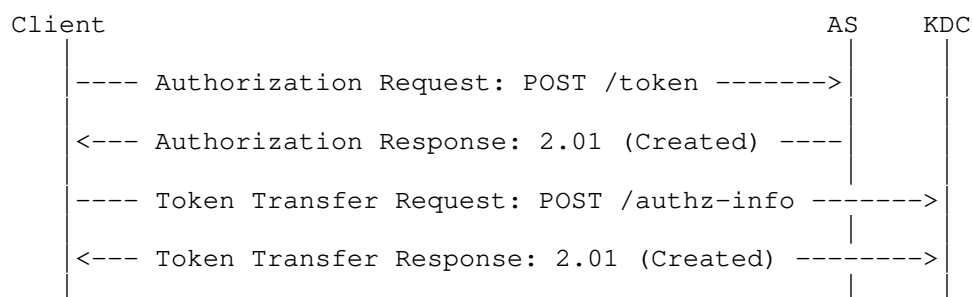


Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

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

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

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

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

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

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

- If the textual format is used, each scope entry is a CBOR array formatted as follows.
 - o As first element, the group name, encoded as a CBOR text string.
 - o Optionally, as second element, the role or CBOR array of roles that the Client wishes to take in the group. This element is optional since roles may have been pre-assigned to the Client, as associated to its verifiable identity credentials. Alternatively, the application may have defined a single, well-known role for the target resource(s) and audience(s).

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

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

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

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

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

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

```

gname = tstr

permissions = uint . bits roles

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

scope_entry = AIF_Generic<gname, permissions>

scope = << [ + scope_entry ] >>

```

Figure 4: Example of scope using the AIF format

```

gname = tstr

role = tstr

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

scope = << [ + scope_entry ] >>

```

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

3.2. Authorization Response

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

In case of successful verification, the Authorization Response sent from the AS to the Client is also defined in Section 5.8.2 of [I-D.ietf-ace-oauth-authz]. Note that the parameter 'expires_in' MAY be omitted if the application defines how the expiration time is communicated to the Client via other means, or if it establishes a default value.

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

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

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

- * a confirmation claim (see for example 'cnf' defined in Section 3.1 of [RFC8747] for CWT);
- * an expiration time claim (see for example 'exp' defined in Section 3.1.4 of [RFC8392] for CWT);
- * a scope claim (see for example 'scope' registered in Section 8.14 of [I-D.ietf-ace-oauth-authz] for CWT).

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

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

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

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

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

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

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

3.3. Token Transferring

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

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

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

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

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

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

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

If the access token contains a role that requires the Client to send its own public key to the KDC when joining the group, the CBOR map MUST include the parameter 'kdcchallenge' defined in Section 3.3.2, specifying a dedicated challenge N_S generated by the KDC.

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

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

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

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

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

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

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

3.3.1. 'sign_info' Parameter

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

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

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

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

When used in the following Token Transfer Response from the KDC (see Section 3.3), the 'sign_info' parameter is a CBOR array of one or more elements. The number of elements is at most the number of groups that the Client has been authorized to join - or to have a more restricted interaction (see above). Each element contains information about signing parameters and about public keys for one or more groups, and is formatted as follows.

- * The first element 'id' is a group name or an array of group names, associated to groups for which the next four elements apply. In the following, each specified group name is referred to as 'gname'.
- * The second element 'sign_alg' is an integer or a text string if the POST request included the 'sign_info' parameter with value the CBOR simple value 'null' (0xf6), and indicates the signature algorithm used in the groups identified by the 'gname' values. It is REQUIRED of the application profiles to define specific values that this parameter can take (REQ3), selected from the set of signing algorithms of the COSE Algorithms registry [COSE.Algorithms].
- * The third element 'sign_parameters' is a CBOR array indicating the parameters of the signature algorithm used in the groups identified by the 'gname' values. Its content depends on the value of 'sign_alg'. It is REQUIRED of the application profiles to define the possible values and structure for the elements of this parameter (REQ4).
- * The fourth element 'sign_key_parameters' is a CBOR array indicating the parameters of the key used with the signature algorithm, in the groups identified by the 'gname' values. Its content depends on the value of 'sign_alg'. It is REQUIRED of the application profiles to define the possible values and structure for the elements of this parameter (REQ5).
- * The fifth element 'pub_key_enc' parameter is either a CBOR integer indicating the encoding of public keys used in the groups identified by the 'gname' values, or has value the CBOR simple value 'null' (0xf6) indicating that the KDC does not act as repository of public keys for group members. Its acceptable integer values are taken from the 'Label' column of the "COSE Header Parameters" registry [COSE.Header.Parameters]. It is REQUIRED of the application profiles to define specific values to use for this parameter, consistently with the acceptable formats of public keys (REQ6).

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

```
sign_info = sign_info_req / sign_info_resp

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

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

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

gname = tstr
```

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

3.3.2. 'kdcchallenge' Parameter

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

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

4. KDC Functionalities

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

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

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

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

4.1. Interface at the KDC

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

It is REQUIRED of the application profiles of this specification to register a Resource Type for the root url-path (REQ10). This Resource Type can be used to discover the correct url to access at the KDC. This Resource Type can also be used at the GROUPNAME sub-resource, to indicate different application profiles for different groups.

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

4.1.1. Operations Supported by Clients

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

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

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

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

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

4.1.2. Error Handling

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

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

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

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

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

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

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

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

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

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

4.2. /ace-group

This resource implements the FETCH handler.

4.2.1. FETCH Handler

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

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

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

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

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

- * 'gid', whose value is encoded as a CBOR array, containing zero or more group identifiers. The handler indicates that those are the identifiers it is sending group names and GROUPNAMEs for. This CBOR array is a subset of the 'gid' array in the FETCH request.
- * 'gname', whose value is encoded as a CBOR array, containing zero or more group names. The elements of this array are encoded as text strings. Each element of index *i* of this CBOR array corresponds to the element of group identifier *i* in the 'gid' array.

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

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

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

4.2.1.1. Retrieve Group Names

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

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

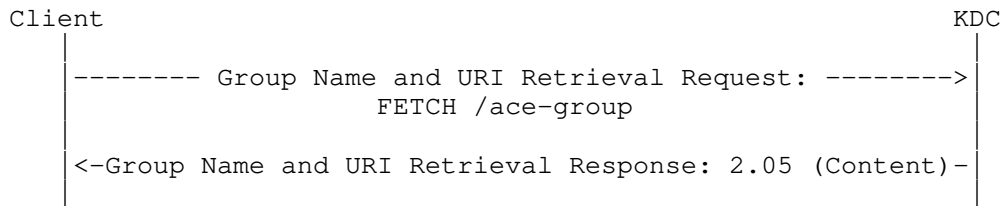


Figure 6: Message Flow of Group Name and URI Retrieval Request-Response

Request:

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

Response:

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

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

4.3. /ace-group/GROUPNAME

This resource implements the POST and GET and handlers.

4.3.1. POST Handler

The POST handler processes the Joining Request sent by a Client to join a group, and returns a Joining Response as successful result of the joining process (see Section 4.3.1.1). At a high level, the POST handler adds the Client to the list of current group members, adds the public key of the Client to the list of the group members' public keys, and returns the symmetric group keying material for the group identified by GROUPNAME.

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

- * 'scope', with value the specific group that the Client is attempting to join, i.e., the group name, and the roles it wishes to have in the group. This value is a CBOR byte string wrapping one scope entry, as defined in Section 3.1.

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

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

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

In particular, the Client indicates that it wishes to retrieve the public keys of all the group members having any of the single roles, or at least all of the roles indicated in any combination of roles. For example, the array `["role1", "role2+role3"]` indicates that the Client wishes to receive the public keys of all group members that have at least `"role1"` or at least both `"role2"` and `"role3"`.

- The third element, namely `'id_filter'`, is an empty CBOR array. That is, when the Joining Request includes a non-Null `'get_pub_keys'` parameter, the Client does not filter public keys based on node identifiers.

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

The CDDL definition [RFC8610] of 'get_pub_keys' is given in Figure 8, using as example encoding: node identifier encoded as a CBOR byte string; role identifier encoded as a CBOR text string, and combination of roles encoded as a CBOR array of roles.

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

```
inclusion_flag = bool

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

id = bstr
id_filter = [ *id ]

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

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

- * 'client_cred', encoded as a CBOR byte string, with value the original binary representation of the Client's public key. This parameter is used if the KDC is managing (collecting from/ distributing to the Client) the public keys of the group members, and if the Client's role in the group will require for it to send messages to one or more group members. It is REQUIRED of the application profiles to define the specific formats that are acceptable to use for encoding public keys in the group (REQ6).
- * 'nonce', encoded as a CBOR byte string, and including a dedicated nonce N_C generated by the Client. This parameter MUST be present if the 'client_cred' parameter is present.

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

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

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

An example of PoP input to compute `'client_cred_verify'` using CBOR encoding is given in Figure 9.

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

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

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

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

If 'control_uri' is specified in the Joining Request, the Client acts as a CoAP server and hosts a resource at this specific URI. The KDC MAY use this URI to send CoAP requests to the Client (acting as CoAP server in this exchange), for example for one-to-one provisioning of new group keying material when performing a group rekeying (see Section 4.8.1.1), or to inform the Client of its removal from the group Section 5.

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

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

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

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

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

PoP input:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Then, the handler performs the following actions.

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

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

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

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

The Joining Response message MUST have Content-Format `application/ace-groupcomm+cbor`. The payload of the response is formatted as a CBOR map, which MUST contain the following fields and values.

- * `'gkty'`, identifying the key type of the `'key'` parameter. The set of values can be found in the "Key Type" column of the "ACE Groupcomm Key Types" registry. Implementations MUST verify that the key type matches the application profile being used, if present, as registered in the "ACE Groupcomm Key Types" registry.
- * `'key'`, containing the keying material for the group communication, or information required to derive it.
- * `'num'`, containing the version number of the keying material for the group communication, formatted as an integer. This is a strictly monotonic increasing field. The application profile MUST define the initial version number (REQ16).

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

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

Figure 10: Key Type Values

The response SHOULD contain the following parameter:

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

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

- * `'ace-groupcomm-profile'`, with value a CBOR integer that MUST be used to uniquely identify the application profile for group communication. Applications of this specification MUST register an application profile identifier and the related value for this parameter in the "ACE Groupcomm Profiles" registry (REQ19).
- * `'pub_keys'`, MUST be present if `'get_pub_keys'` was present in the request, otherwise it MUST NOT be present. This parameter is a CBOR array specifying the public keys of the group members, i.e., of all of them or of the ones selected according to the `'get_pub_keys'` parameter in the request. In particular, each element of the array is a CBOR byte string, with value the original binary representation of a group member's public key. It is REQUIRED of the application profiles to define the specific formats of public keys that are acceptable to use in the group (REQ6).
- * `'peer_roles'`, MUST be present if `'pub_keys'` is also present, otherwise it MUST NOT be present. This parameter is a CBOR array of n elements, with n the number of public keys included in the `'pub_keys'` parameter (at most the number of members in the group). The i -th element of the array specifies the role (or CBOR array of roles) that the group member associated to the i -th public key in `'pub_keys'` has in the group. In particular, each array element is encoded as the role element of a scope entry, as defined in Section 3.1.

- * `'peer_identifiers'`, MUST be present if `'pub_keys'` is also present, otherwise it MUST NOT be present. This parameter is a CBOR array of `n` elements, with `n` the number of public keys included in the `'pub_keys'` parameter (at most the number of members in the group). The `i`-th element of the array specifies the node identifier that the group member associated to the `i`-th public key in `'pub_keys'` has in the group. In particular, the `i`-th array element is encoded as a CBOR byte string, with value the node identifier of the group member.
- * `'group_policies'`, with value a CBOR map, whose entries specify how the group handles specific management aspects. These include, for instance, approaches to achieve synchronization of sequence numbers among group members. The elements of this field are registered in the "ACE Groupcomm Policies" registry. This specification defines the three elements "Sequence Number Synchronization Methods", "Key Update Check Interval" and "Expiration Delta", which are summarized in Figure 11. Application profiles that build on this document MUST specify the exact content format and default value of included map entries (REQ20).

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

Figure 11: ACE Groupcomm Policies

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

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

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

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

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

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

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

Figure 12: ACE Groupcomm Rekeying Schemes

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

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

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

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

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

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

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

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

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

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

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

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

4.3.1.1. Join the Group

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

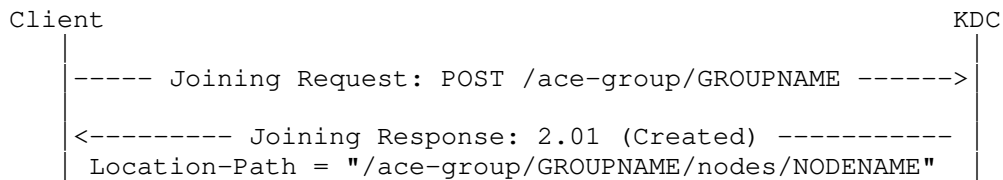


Figure 13: Message Flow of the Joining Exchange

Request:

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

Response:

```
Header: Created (Code=2.01)
Content-Format: "application/ace-groupcomm+cbor"
Location-Path: "kdc.example.com"
Location-Path: "g1"
Location-Path: "nodes"
Location-Path: "c101"
Payload (in CBOR diagnostic notation,
        with KEY being a CBOR byte strings):
{ "gkty": 13, "key": KEY, "num": 12, "exp": 1609459200,
  "pub_keys": [ PUB_KEY1, PUB_KEY2 ],
  "peer_roles": ["sender", ["sender", "receiver"]],
  "peer_identifiers": [ ID1, ID2 ] }
```

Figure 14: Example of First Exchange for Group Joining

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

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

To join the group, the Client sends a CoAP POST request to the /ace-group/GROUPNAME endpoint at the KDC, where GROUPNAME is the group name of the group to join, formatted as specified in Section 4.3.1. This group name is the same as in the scope entry corresponding to

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

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

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

4.3.2. GET Handler

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

The handler expects a GET request.

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

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

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

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

4.3.2.1. Retrieve Group Keying Material

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

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

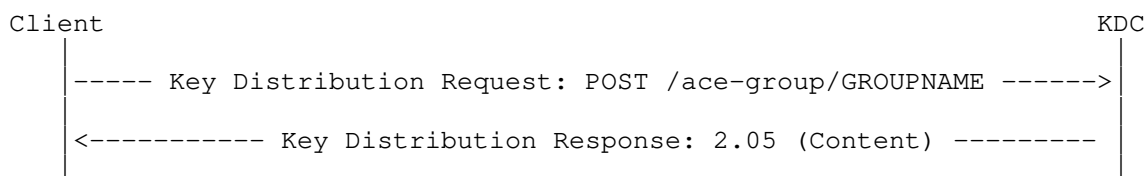


Figure 15: Message Flow of Key Distribution Request-Response

Request:

```

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

Response:

```

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

Figure 16: Example of Key Distribution Request-Response

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

This resource implements the GET and FETCH handlers.

4.4.1. FETCH Handler

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

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

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

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

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

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

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

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

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

The handler identifies the public keys of the current group members for which either:

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

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

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

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

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

Request:

```
Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "pub-key"
Content-Format: "application/ace-groupcomm+cbor"
Payload:
  { "get_pub_keys": [true, [], [ ID3 ]] }
```

Response:

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

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

4.4.2. GET Handler

The handler expects a GET request.

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

4.4.2.1. Retrieve All Public Keys in the Group

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

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

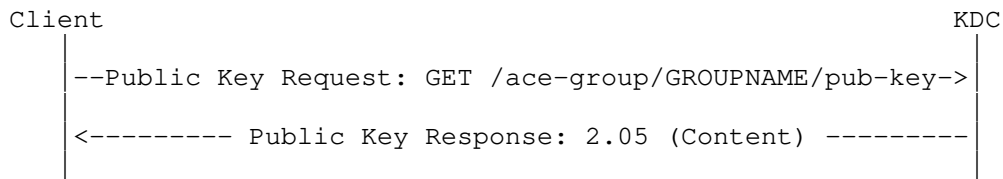


Figure 19: Message Flow of Public Key Exchange to Request the Public Keys of all the Group Members

Request:

```

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

Response:

```

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

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

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

This resource implements a GET handler.

4.5.1. GET Handler

The handler expects a GET request.

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

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

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

4.5.1.1. Retrieve the KDC's Public Key

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

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

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

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

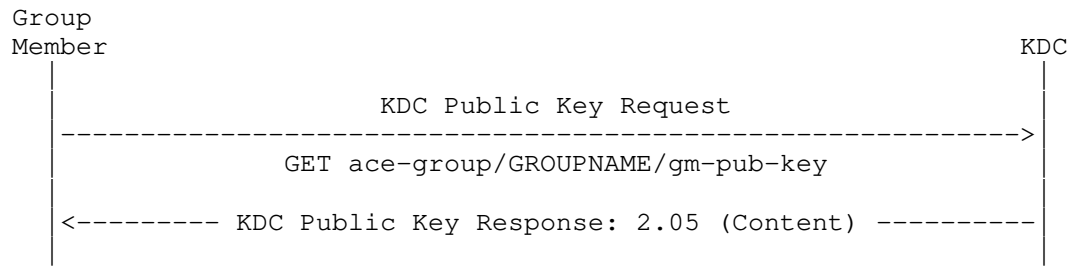


Figure 21: Message Flow of KDC Public Key Request-Response

Request:

```

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

Response:

```

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

Figure 22: Example of KDC Public Key Request-Response

4.6. /ace-group/GROUPNAME/policies

This resource implements the GET handler.

4.6.1. GET Handler

The handler expects a GET request.

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

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

The specific format and meaning of group policies MUST be specified in the application profile (REQ20).

4.6.1.1. Retrieve the Group Policies

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

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

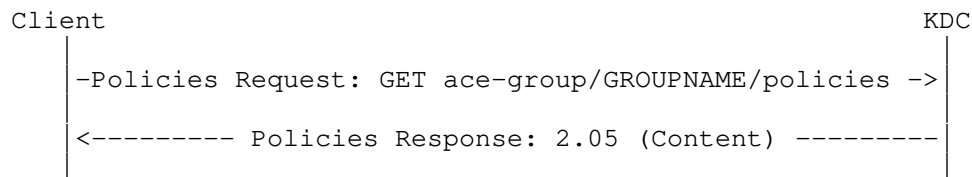


Figure 23: Message Flow of Policies Request-Response

Request:

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

Response:

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

Figure 24: Example of Policies Request-Response

4.7. /ace-group/GROUPNAME/num

This resource implements the GET handler.

4.7.1. GET Handler

The handler expects a GET request.

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

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

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

4.7.1.1. Retrieve the Keying Material Version

A node in the group can contact the KDC to request information about the version number of the symmetric group keying material, by sending a CoAP GET request to the `/ace-group/GROUPNAME/num` endpoint at the KDC, where `GROUENAME` is the group name, formatted as defined in Section 4.7.1. In particular, the version is incremented by the KDC every time the group keying material is renewed, before it's distributed to the group members.

Figure 25 gives an overview of the exchange described above, while Figure 26 shows an example.

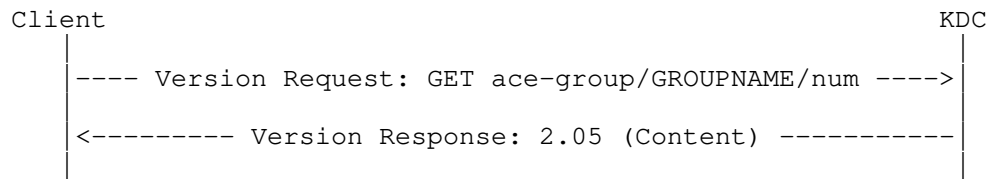


Figure 25: Message Flow of Version Request-Response

Request:

```

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

Response:

```

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

Figure 26: Example of Version Request-Response

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

This resource implements the GET, PUT and DELETE handlers.

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

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

4.8.1. GET Handler

The handler expects a GET request.

If all verifications succeed, the handler replies with a 2.05 (Content) response containing both the group keying material and the individual keying material for the Client, or information enabling the Client to derive it. The payload of the response is formatted as a CBOR map. The format for the group keying material is the same as defined in the response of Section 4.3.2. The specific format of individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ27) and registered in Section 11.7.

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

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

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

4.8.1.1. Retrieve Group and Individual Keying Material

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

- * Upon expiration of the keying material, according to what indicated by the KDC with the 'exp' parameter in a Joining Response, or to a pre-configured value.
- * Upon receiving a notification of revoked/renewed keying material from the KDC, possibly as part of an update of the keying material (rekeying) triggered by the KDC.
- * Upon receiving messages from other group members without being able to retrieve the keying material to correctly decrypt them. This may be due to rekeying messages previously sent by the KDC, that the Client was not able to receive or decrypt.

In either case, if it wants to continue participating in the group communication, the node has to request the latest keying material from the KDC. To this end, the Client sends a CoAP GET request to the /ace-group/GROUPNAME/nodes/NODENAME endpoint at the KDC, formatted as specified in Section 4.8.1.

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

It is application dependent and pertaining to the particular message exchange (e.g., [I-D.ietf-core-oscure-groupcomm]) to set up these policies for instructing Clients to retain incoming messages and for how long (OPT11). This allows Clients to possibly decrypt such messages after getting updated keying material, rather than just consider them non valid messages to discard right away.

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

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

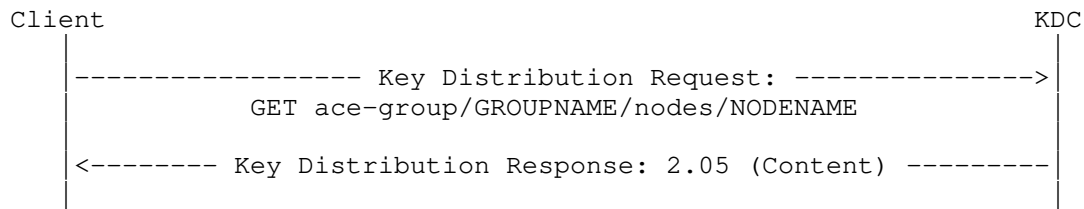


Figure 27: Message Flow of Key Distribution Request-Response

Request:

```

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

```

Response:

```

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

```

Figure 28: Example of Key Distribution Request-Response

4.8.2. PUT Handler

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

The handler expects a PUT request with empty payload.

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

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

If all verifications succeed, the handler reply with a 2.05 (Content) response containing newly generated, individual keying material for the Client. The payload of the response is formatted as a CBOR map. The specific format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label, MUST be specified in the application profile (REQ27) and registered in Section 11.7.

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

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

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

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

4.8.2.1. Request to Change Individual Keying Material

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

To this end, the Client performs a Key Renewal Request/Response exchange with the KDC, i.e., it sends a CoAP PUT request to the /ace-group/GROUPNAME/nodes/NODENAME endpoint at the KDC, where GROUPNAME is the group name and NODENAME is its node name, and formatted as defined in Section 4.8.1.

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

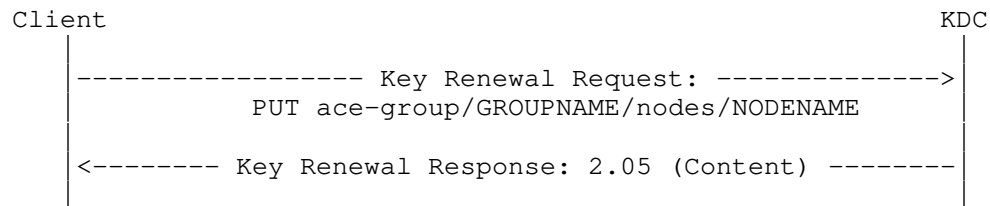


Figure 29: Message Flow of Key Renewal Request-Response

Request:

```

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

Response:

```

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

Figure 30: Example of Key Renewal Request-Response

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

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

4.8.3. DELETE Handler

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

The handler expects a DELETE request with empty payload.

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

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

4.8.3.1. Leave the Group

A Client can actively request to leave the group. In this case, the Client sends a CoAP DELETE request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the KDC, where GROUPNAME is the group name and NODENAME is its node name, formatted as defined in Section 4.8.3

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

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

This resource implements the POST handler.

4.9.1. POST Handler

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

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

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

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

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

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

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

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

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

4.9.1.1. Uploading a New Public Key

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

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

The request is formatted as specified in Section 4.9.1.

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

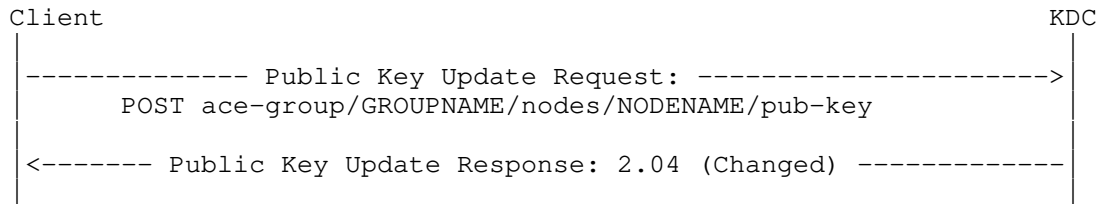


Figure 31: Message Flow of Public Key Update Request-Response

Request:

```

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

Response:

```

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

Figure 32: Example of Public Key Update Request-Response

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

5. Removal of a Group Member

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

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

In either case, the KDC performs the following actions.

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

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

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

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

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

6. Group Rekeying Process

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

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

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

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

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

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

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

6.1. Point-to-Point Group Rekeying

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

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

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

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

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

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

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

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

6.2. One-to-Many Group Rekeying

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.2.1. Protection of Rekeying Messages

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

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

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

First, the KDC computes a COSE_Encrypt0 object as follows.

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

The KDC considers the following values.

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

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

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

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

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

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

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

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

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

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

7. Extended Scope Format

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

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

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

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

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

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

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

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

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

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

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

```

gname = tstr

permissions = uint . bits roles

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

scope_entry = AIF_Generic<gname, permissions>

scope = << [ + scope_entry ] >>

semantics = int

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

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

```

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

```

gname = tstr

role = tstr

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

scope = << [ + scope_entry ] >>

semantics = int

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

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

```

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

8. ACE Groupcomm Parameters

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

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

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

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

Figure 35: ACE Groupcomm Parameters

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

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

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

A Client MUST support the following parameters.

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

A Client SHOULD support the following parameter.

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

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

- * 'error', 'error_description'.

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

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

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

9. ACE Groupcomm Error Identifiers

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

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

Figure 36: ACE Groupcomm Error Identifiers

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

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

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

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

10. Security Considerations

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

Furthermore, the following security considerations apply.

10.1. Secure Communication in the Group

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

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

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

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

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

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

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

10.2. Update of Group Keying Material

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

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

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

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

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

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

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

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

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

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

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

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

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

10.2.1. Misalignment of Group Keying Material

A group member can receive a message shortly after the group has been rekeyed, and new keying material has been distributed by the KDC (see Section 6). In the following two cases, this may result in misaligned keying material between the group members.

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

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

10.3. Block-Wise Considerations

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

Compared to a scenario where the transfer does not use Block-Wise, depending on how fast the group keying material is changed, the group members might consume a larger amount of the network bandwidth by repeatedly resending the same blocks, which might be problematic.

11. IANA Considerations

This document has the following actions for IANA.

11.1. Media Type Registrations

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

Type name: application

Subtype name: ace-groupcomm+cbor

Required parameters: N/A

Optional parameters: N/A

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

Security considerations: See Section 10 of this document.

Interoperability considerations: n/a

Published specification: [this document]

Applications that use this media type: The type is used by Authorization Servers, Clients and Resource Servers that support the ACE groupcomm framework as specified in [this document].

Fragment identifier considerations: N/A

Additional information: N/A

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

Intended usage: COMMON

Restrictions on usage: None

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

Change controller: IESG

11.2. CoAP Content-Formats

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

Media Type: application/ace-groupcomm+cbor

Encoding: -

ID: TBD

Reference: [this document]

11.3. OAuth Parameters

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

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

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

11.4. OAuth Parameters CBOR Mappings

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

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

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

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

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

- * Attribute Value: ace.group

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

11.6. CBOR Tags

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

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

11.7. ACE Groupcomm Parameters

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

The columns of this registry are:

- * Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- * CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.
- * CBOR Type: This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.
- * Reference: This contains a pointer to the public specification for the item.

This registry has been initially populated by the values in Section 8. The Reference column for all of these entries refers to sections of this document.

11.8. ACE Groupcomm Key Types

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

The columns of this registry are:

- * Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- * Key Type Value: This is the value used to identify the keying material. These values MUST be unique. The value can be a positive integer, a negative integer, or a text string.
- * Profile: This field may contain one or more descriptive strings of application profiles to be used with this item. The values should be taken from the Name column of the "ACE Groupcomm Profiles" registry.
- * Description: This field contains a brief description of the keying material.
- * References: This contains a pointer to the public specification for the format of the keying material, if one exists.

This registry has been initially populated by the values in Figure 10. The specification column for all of these entries will be this document.

11.9. ACE Groupcomm Profiles

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

The columns of this registry are:

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

- * Description: Text giving an overview of the application profile and the context it is developed for.
- * CBOR Value: CBOR abbreviation for the name of this application profile. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
- * Reference: This contains a pointer to the public specification of the abbreviation for this application profile, if one exists.

11.10. ACE Groupcomm Policies

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

The columns of this registry are:

- * Name: The name of the group communication policy.
- * CBOR label: The value to be used to identify this group communication policy. Key map labels MUST be unique. The label can be a positive integer, a negative integer or a string. Integer values between 0 and 255 and strings of length 1 are designated as Standards Track Document required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values greater than 65535 and strings of length greater than 2 are designated as expert review. Integer values less than -65536 are marked as private use.
- * CBOR type: the CBOR type used to encode the value of this group communication policy.
- * Description: This field contains a brief description for this group communication policy.
- * Reference: This field contains a pointer to the public specification providing the format of the group communication policy, if one exists.

This registry will be initially populated by the values in Figure 11.

11.11. Sequence Number Synchronization Methods

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

The columns of this registry are:

- * Name: The name of the sequence number synchronization method.
- * Value: The value to be used to identify this sequence number synchronization method.
- * Description: This field contains a brief description for this sequence number synchronization method.
- * Reference: This field contains a pointer to the public specification describing the sequence number synchronization method.

11.12. ACE Scope Semantics

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

The columns of this registry are:

- * Value: The value to be used to identify this scope semantics. The value MUST be unique. The value can be a positive integer or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as expert review. Integer values less than -65536 are marked as private use.
- * Description: This field contains a brief description of the scope semantics.

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

11.13. ACE Groupcomm Errors

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

The columns of this registry are:

- * Value: The value to be used to identify the error. The value MUST be unique. The value can be a positive integer or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as expert review. Integer values less than -65536 are marked as private use.
- * Description: This field contains a brief description of the error.
- * Reference: This field contains a pointer to the public specification defining the error, if one exists.

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

11.14. ACE Groupcomm Rekeying Schemes

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

The columns of this registry are:

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

from 256 to 65535 are designated as Specification Required.
Integer values greater than 65535 are designated as expert review.
Integer values less than -65536 are marked as private use.

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

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

11.15. Expert Review Instructions

The IANA Registries established in this document are defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- * Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments, code points in other ranges should not be assigned for testing.
- * Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

- * Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

12. References

12.1. Normative References

- [COSE.Algorithms]
IANA, "COSE Algorithms",
<<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.
- [COSE.Header.Parameters]
IANA, "COSE Header Parameters",
<<https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>>.
- [I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", Work in Progress, Internet-Draft, draft-ietf-ace-aif-03, 24 June 2021,
<<https://www.ietf.org/archive/id/draft-ietf-ace-aif-03.txt>>.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-46, 8 November 2021,
<<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-13, 25 October 2021,
<<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-13.txt>>.

- [I-D.ietf-cose-countersign]
Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.
- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.
- [I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

12.2. Informative References

- [I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.
- [I-D.ietf-ace-mqtt-tls-profile]
Sengul, C. and A. Kirby, "Message Queuing Telemetry Transport (MQTT)-TLS profile of Authentication and Authorization for Constrained Environments (ACE) Framework", Work in Progress, Internet-Draft, draft-ietf-ace-mqtt-tls-profile-13, 23 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-mqtt-tls-profile-13.txt>>.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

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

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

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

Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-05, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-05.txt>>.

[I-D.tiloca-core-oscore-discovery]

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

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

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

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

- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Requirements on Application Profiles

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

A.1. Mandatory-to-Address Requirements

- * REQ1: Specify the format and encoding of 'scope'. This includes defining the set of possible roles and their identifiers, as well as the corresponding encoding to use in the scope entries according to the used scope format (see Section 3.1).
- * REQ2: If the AIF format of 'scope' is used, register its specific instance of "Toid" and "Tperm", as well as the corresponding Media Type and Content-Format, as per the guidelines in [I-D.ietf-ace-aif].
- * REQ3: If used, specify the acceptable values for 'sign_alg' (see Section 3.3).

- * REQ4: If used, specify the acceptable values for 'sign_parameters' (see Section 3.3).
- * REQ5: If used, specify the acceptable values for 'sign_key_parameters' (see Section 3.3).
- * REQ6: Specify the acceptable formats for encoding public keys and, if used, the acceptable values for 'pub_key_enc' (see Section 3.3).
- * REQ7: If the value of the GROUPNAME URI path and the group name in the access token scope (gname in Section 3.2) are not required to coincide, specify the mechanism to map the GROUPNAME value in the URI to the group name (see Section 4.1).
- * REQ8: Define whether the KDC has a public key and if this has to be provided through the 'kdc_cred' parameter, see Section 4.3.1.
- * REQ9: Specify if any part of the KDC interface as defined in this document is not supported by the KDC (see Section 4.1).
- * REQ10: Register a Resource Type for the root url-path, which is used to discover the correct url to access at the KDC (see Section 4.1).
- * REQ11: Define what specific actions (e.g., CoAP methods) are allowed on each resource provided by the KDC interface, depending on whether the Client is a current group member; the roles that a Client is authorized to take as per the obtained access token (see Section 3.1); and the roles that the Client has as current group member.
- * REQ12: Categorize possible newly defined operations for Clients into primary operations expected to be minimally supported and secondary operations, and provide accompanying considerations (see Section 4.1.1).
- * REQ13: Specify the encoding of group identifier (see Section 4.2.1).
- * REQ14: Specify the approaches used to compute and verify the PoP evidence to include in 'client_cred_verify', and which of those approaches is used in which case (see Section 4.3.1).
- * REQ15: Specify how the nonce N_S is generated, if the token is not provided to the KDC through the Token Transfer Request to the authz-info endpoint (e.g., if it is used directly to validate TLS instead).

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

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

A.2. Optional-to-Address Requirements

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

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

Appendix B. Extensibility for Future COSE Algorithms

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

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

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

B.1. Format of 'sign_info_entry'

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

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

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

Figure 37: 'sign_info_entry' with general format

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. Version -14 to -15

- * Fixed nits.

C.2. Version -13 to -14

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

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

C.3. Version -05 to -13

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

C.4. Version -04 to -05

- * Updated uppercase/lowercase URI segments for KDC resources.
- * Supporting single Access Token for multiple groups/topics.
- * Added 'control_uri' parameter in the Joining Request.
- * Added 'peer_roles' parameter to support legal requesters/responders.
- * Clarification on stopping using owned keying material.
- * Clarification on different reasons for processing failures, related policies, and requirement OPT11.
- * Added a KDC sub-resource for group members to upload a new public key.
- * Possible group rekeying following an individual Key Renewal Request.
- * Clarified meaning of requirement REQ3; added requirement OPT12.
- * Editorial improvements.

C.5. Version -03 to -04

- * Revised RESTful interface, as to methods and parameters.
- * Extended processing of joining request, as to check/retrieval of public keys.
- * Revised and extended profile requirements.
- * Clarified specific usage of parameters related to signature algorithms/keys.
- * Included general content previously in draft-ietf-ace-key-groupcomm-oscore
- * Registration of media type and content format application/ace-group+cbor
- * Editorial improvements.

C.6. Version -02 to -03

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

- * Restructured KDC interface, with new possible operations (Section 4).
- * Client PoP signature for the Joining Request upon joining (Section 4.1.2.1).
- * Revised text on group member removal (Section 5).
- * Added more profile requirements (Appendix A).

C.7. Version -01 to -02

- * Editorial fixes.
- * Distinction between transport profile and application profile (Section 1.1).
- * New parameters 'sign_info' and 'pub_key_enc' to negotiate parameter values for signature algorithm and signature keys (Section 3.3).
- * New parameter 'type' to distinguish different Key Distribution Request messages (Section 4.1).
- * New parameter 'client_cred_verify' in the Key Distribution Request to convey a Client signature (Section 4.1).
- * Encoding of 'pub_keys_repos' (Section 4.1).
- * Encoding of 'mgt_key_material' (Section 4.1).
- * Improved description on retrieval of new or updated keying material (Section 6).
- * Encoding of 'get_pub_keys' in Public Key Request (Section 7.1).
- * Extended security considerations (Sections 10.1 and 10.2).
- * New "ACE Public Key Encoding" IANA registry (Section 11.2).
- * New "ACE Groupcomm Parameters" IANA registry (Section 11.3), populated with the entries in Section 8.
- * New "Ace Groupcomm Request Type" IANA registry (Section 11.4), populated with the values in Section 9.

- * New "ACE Groupcomm Policy" IANA registry (Section 11.7) populated with two entries "Sequence Number Synchronization Method" and "Key Update Check Interval" (Section 4.2).
- * Improved list of requirements for application profiles (Appendix A).

C.8. Version -00 to -01

- * Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).
- * Defined error handling on the KDC (Sections 4.2 and 6.2).
- * Updated format of the Key Distribution Response as a whole (Section 4.2).
- * Generalized format of 'pub_keys' in the Key Distribution Response (Section 4.2).
- * Defined format for the message to request leaving the group (Section 5.2).
- * Renewal of individual keying material and methods for group rekeying initiated by the KDC (Section 6).
- * CBOR type for node identifiers in 'get_pub_keys' (Section 7.1).
- * Added section on parameter identifiers and their CBOR keys (Section 8).
- * Added request types for requests to a Join Response (Section 9).
- * Extended security considerations (Section 10).
- * New IANA registries "ACE Groupcomm Key registry", "ACE Groupcomm Profile registry", "ACE Groupcomm Policy registry" and "Sequence Number Synchronization Method registry" (Section 11).
- * Added appendix about requirements for application profiles of ACE on group communication (Appendix A).

Acknowledgments

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

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

Authors' Addresses

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

Email: francesca.palombini@ericsson.com

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

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 30 October 2022

M. Tiloca
RISE AB
J. Park
Universitaet Duisburg-Essen
F. Palombini
Ericsson AB
28 April 2022

Key Management for OSCORE Groups in ACE
draft-ietf-ace-key-groupcomm-oscore-14

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 30 October 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	5
2. Protocol Overview	6
3. Format of Scope	8
4. Authentication Credentials	10
5. Authorization to Join a Group	12
5.1. Authorization Request	12
5.2. Authorization Response	13
5.3. Token Transferring	13
5.3.1. 'ecdh_info' Parameter	16
5.3.2. 'kdc_dh_creds' Parameter	18
6. Group Joining	20
6.1. Send the Joining Request	20
6.1.1. Value of the N_S Challenge	22
6.2. Receive the Joining Request	22
6.2.1. Follow-up to a 4.00 (Bad Request) Error Response	25
6.3. Send the Joining Response	26
6.4. Receive the Joining Response	32
7. Overview of the Group Rekeying Process	34
7.1. Stale OSCORE Sender IDs	35
8. Interface at the Group Manager	37
8.1. ace-group/GROUPNAME/active	37
8.1.1. GET Handler	37
8.2. ace-group/GROUPNAME/verif-data	38
8.2.1. GET Handler	38
8.3. ace-group/GROUPNAME/stale-sids	38
8.3.1. FETCH Handler	38
8.4. Admitted Methods	39
8.4.1. Signature Verifiers	40
8.5. Operations Supported by Clients	41
9. Additional Interactions with the Group Manager	41
9.1. Retrieve Updated Keying Material	42
9.1.1. Get Group Keying Material	42
9.1.2. Get Group Keying Material and OSCORE Sender ID	42
9.2. Request to Change Individual Keying Material	43
9.3. Retrieve Authentication Credentials of Group Members	45
9.4. Upload a New Authentication Credential	45

9.5.	Retrieve the Group Manager's Authentication Credential	47
9.6.	Retrieve Signature Verification Data	48
9.7.	Retrieve the Group Policies	50
9.8.	Retrieve the Keying Material Version	50
9.9.	Retrieve the Group Status	50
9.10.	Retrieve Group Names	51
9.11.	Leave the Group	54
10.	Removal of a Group Member	54
11.	Group Rekeying Process	56
11.1.	Sending Rekeying Messages	58
11.2.	Receiving Rekeying Messages	60
11.3.	Missed Rekeying Instances	61
11.3.1.	Retrieve Stale Sender IDs	63
12.	ACE Groupcomm Parameters	65
13.	ACE Groupcomm Error Identifiers	67
14.	Default Values for Group Configuration Parameters	68
14.1.	Common	68
14.2.	Group Mode	69
14.3.	Pairwise Mode	70
15.	Security Considerations	71
15.1.	Management of OSCORE Groups	71
15.2.	Size of Nonces as Proof-of-Possession Challenge	72
15.3.	Reusage of Nonces for Proof-of-Possession Input	73
16.	IANA Considerations	74
16.1.	OAuth Parameters	74
16.2.	OAuth Parameters CBOR Mappings	74
16.3.	ACE Groupcomm Parameters	75
16.4.	ACE Groupcomm Key Types	76
16.5.	ACE Groupcomm Profiles	76
16.6.	OSCORE Security Context Parameters	76
16.7.	TLS Exporter Labels	78
16.8.	AIF	79
16.9.	CoAP Content-Format	79
16.10.	Group OSCORE Roles	80
16.11.	CoRE Resource Type	80
16.12.	ACE Scope Semantics	81
16.13.	ACE Groupcomm Errors	81
16.14.	Expert Review Instructions	82
17.	References	82
17.1.	Normative References	82
17.2.	Informative References	86
Appendix A.	Profile Requirements	88
A.1.	Mandatory-to-Address Requirements	88
A.2.	Optional-to-Address Requirements	91
Appendix B.	Extensibility for Future COSE Algorithms	92
B.1.	Format of 'ecdh_info_entry'	93
B.2.	Format of 'key'	94
Appendix C.	Document Updates	95

C.1.	Version -13 to -14	95
C.2.	Version -12 to -13	95
C.3.	Version -11 to -12	95
C.4.	Version -10 to -11	96
C.5.	Version -09 to -10	97
C.6.	Version -08 to -09	97
C.7.	Version -07 to -08	98
C.8.	Version -06 to -07	98
C.9.	Version -05 to -06	99
C.10.	Version -04 to -05	99
C.11.	Version -03 to -04	100
C.12.	Version -02 to -03	100
C.13.	Version -01 to -02	101
C.14.	Version -00 to -01	102
Acknowledgments		102
Authors' Addresses		102

1. Introduction

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

As described in [I-D.ietf-core-oscore-groupcomm], Group OSCORE is used to protect CoAP group communication [I-D.ietf-core-groupcomm-bis], which can employ, for example, IP multicast as underlying data transport. This relies on a Group Manager, which is responsible for managing an OSCORE group and enables the group members to exchange CoAP messages secured with Group OSCORE. The Group Manager can be responsible for multiple groups, coordinates the joining process of new group members, and is entrusted with the distribution and renewal of group keying material.

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

1.1. Terminology

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

Readers are expected to be familiar with:

- * The terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] and in the Authorization Information Format (AIF) [I-D.ietf-ace-aif] to express authorization information. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).
- * The terms and concept related to the message formats and processing specified in [I-D.ietf-ace-key-groupcomm], for provisioning and renewing keying material in group communication scenarios.
- * The terms and concepts described in CBOR [RFC8949] and COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].
- * The terms and concepts described in CoAP [RFC7252] and group communication for CoAP [I-D.ietf-core-groupcomm-bis]. Unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS, and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".
- * The terms and concepts for protection and processing of CoAP messages through OSCORE [RFC8613] and through Group OSCORE [I-D.ietf-core-oscore-groupcomm] in group communication scenarios. These especially include:
 - Group Manager, as the entity responsible for a set of groups where communications are secured with Group OSCORE. In this document, the Group Manager acts as Resource Server.

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

Additionally, this document makes use of the following terminology.

- * Requester: member of an OSCORE group that sends request messages to other members of the group.
- * Responder: member of an OSCORE group that receives request messages from other members of the group. A responder may reply back, by sending a response message to the requester which has sent the request message.
- * Monitor: member of an OSCORE group that is configured as responder and never replies back to requesters after receiving request messages. This corresponds to the term "silent server" used in [I-D.ietf-core-oscore-groupcomm].
- * Signature verifier: entity external to the OSCORE group and intended to verify the signature of messages exchanged in the group (see Sections 3.1 and 8.5 of [I-D.ietf-core-oscore-groupcomm]). An authorized signature verifier does not join the OSCORE group as an actual member, yet it can retrieve the authentication credentials of the current group members from the Group Manager.
- * Signature-only group: an OSCORE group that uses only the group mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]).
- * Pairwise-only group: an OSCORE group that uses only the pairwise mode (see Section 9 of [I-D.ietf-core-oscore-groupcomm]).

2. Protocol Overview

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

This document describes how to use [I-D.ietf-ace-key-groupcomm] and [I-D.ietf-ace-oauth-authz] to perform a number of authentication, authorization and key distribution actions as overviewed in Section 2 of [I-D.ietf-ace-key-groupcomm], when the considered group is specifically an OSCORE group.

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

- * The node wishing to join the OSCORE group, i.e., the joining node, is the Client.
- * The Group Manager is the Key Distribution Center (KDC), acting as a Resource Server.
- * The Authorization Server associated with the Group Manager is the AS.

A node performs the steps described in Sections 3 and 4.3.1.1 of [I-D.ietf-ace-key-groupcomm] in order to obtain an authorization for joining an OSCORE group and then to join that group. The format and processing of messages exchanged during such steps are further specified in Section 5 and Section 6 of this document.

All communications between the involved entities MUST be secured.

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

With respect to what is defined in [I-D.ietf-ace-key-groupcomm]:

- * The interface provided by the Group Manager extends the original interface defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm] for the KDC, as specified in Section 8 of this document.
- * In addition to those defined in Section 8 of [I-D.ietf-ace-key-groupcomm], additional parameters are defined in this document and summarized in Section 12.
- * In addition to those defined in Section 9 of [I-D.ietf-ace-key-groupcomm], additional error identifiers are defined in this document and summarized in Section 13.

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

3. Format of Scope

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

To this end, this profile uses the Authorization Information Format (AIF) [I-D.ietf-ace-aif]. In particular, with reference to the generic AIF model

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

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

Furthermore, this document defines the new AIF specific data model AIF-OSCORE-GROUPCOMM, that this profile MUST use to format and encode scope entries.

In particular, the following holds for each scope entry.

- * The object identifier ("Toid") is specialized as a CBOR item specifying the name of the groups pertaining to the scope entry.
- * The permission set ("Tperm") is specialized as a CBOR unsigned integer with value R, specifying the permissions that the Client wishes to have in the groups indicated by "Toid".

More specifically, the following applies when, as defined in this document, a scope entry includes as set of permissions the set of roles to take in an OSCORE group.

- * The object identifier ("Toid") is a CBOR text string, specifying the group name for the scope entry.
- * The permission set ("Tperm") is a CBOR unsigned integer with value R, specifying the role(s) that the Client wishes to take in the group (REQ1). The value R is computed as follows.
 - Each role in the permission set is converted into the corresponding numeric identifier X from the "Value" column of the "Group OSCORE Roles" registry, for which this document defines the entries in Figure 1.

- The set of N numbers is converted into the single value R, by taking two to the power of each numeric identifier X₁, X₂, ..., X_N, and then computing the inclusive OR of the binary representations of all the power values.

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

Figure 1: Numeric identifier of roles in an OSCORE group

The following CDDL [RFC8610] notation defines a scope entry that uses the AIF-OSCORE-GROUPCOMM data model and expresses a set of Group OSCORE roles from those in Figure 1.

```

AIF-OSCORE-GROUPCOMM = AIF-Generic<oscore-gname, oscore-gperm>

oscore-gname = tstr ; Group name
oscore-gperm = uint . bits group-oscore-roles

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

scope_entry = [oscore-gname, oscore-gperm]
```

Future specifications that define new Group OSCORE roles MUST register a corresponding numeric identifier in the "Group OSCORE Roles" registry defined in Section 16.10 of this document.

Note that the value 0 is not available to use as numeric identifier to specify a Group OSCORE role. It follows that, when expressing Group OSCORE roles to take in a group as per this document, a scope entry has the least significant bit of "Tperm" always set to 0.

This is an explicit feature of the AIF-OSCORE-GROUPCOMM data model. That is, for each scope entry, the least significant bit of "Tperm" set to 0 explicitly identifies the scope entry as exactly expressing a set of Group OSCORE roles ("Tperm"), pertaining to a single group whose name is specified by the string literal in "Toid".

Instead, by relying on the same AIF-OSCORE-GROUPCOMM data model, [I-D.ietf-ace-oscore-gm-admin] defines the format of scope entries for Administrator Clients that wish to access an admin interface at the Group Manager. In such scope entries, the least significant bit of "Tperm" is always set to 1.

4. Authentication Credentials

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

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

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

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

- * The joining node is going to join the group exclusively as monitor, i.e., it is not going to send messages to the group. In this case, the joining node is not required to provide its own authentication credential to the Group Manager, which thus does not have to perform any check related to the format of the authentication credential, to a signature or ECDH algorithm, and to possible parameters associated with the algorithm and the public key. In case the joining node still provides an authentication credential in the 'client_cred' parameter of the Joining Request (see Section 6.1), the Group Manager silently ignores that parameter, as well as the related parameters 'cnonce' and 'client_cred_verify'.

- * The Group Manager already acquired the authentication credential of the joining node during a past joining process. In this case, the joining node MAY choose not to provide again its own authentication credential to the Group Manager, in order to limit the size of the Joining Request. The joining node MUST provide its own authentication credential again if it has provided the Group Manager with multiple authentication credentials during past joining processes, intended for different OSCORE groups. If the joining node provides its own authentication credential, the Group Manager performs consistency checks as per Section 6.2 and, in case of success, considers it as the authentication credential associated with the joining node in the OSCORE group.
- * The joining node and the Group Manager use an asymmetric proof-of-possession key to establish a secure communication association. Then, two cases can occur.
 1. When establishing the secure communication association, the Group Manager obtained from the joining node the joining node's authentication credential, in the format used in the OSCORE group and including the asymmetric proof-of-possession key as public key. Also, such authentication credential and the proof-of-possession key are compatible with the signature or ECDH algorithm, and possible associated parameters used in the OSCORE group.

In this case, the Group Manager considers the authentication credential as the one associated with the joining node in the OSCORE group. If the joining node is aware that the authentication credential and the public key included thereof are also valid for the OSCORE group, then the joining node MAY choose to not provide again its own authentication credential to the Group Manager.

The joining node MUST provide again its own authentication credential if it has provided the Group Manager with multiple authentication credentials during past joining processes, intended for different OSCORE groups. If the joining node provides its own authentication credential in the 'client_cred' parameter of the Joining Request (see Section 6.1), the Group Manager performs consistency checks as per Section 6.2 and, in case of success, considers it as the authentication credential associated with the joining node in the OSCORE group.

2. The authentication credential is not in the format used in the OSCORE group, or else the authentication credential and the proof-of-possession key included as public key are not compatible with the signature or ECDH algorithm, and possible associated parameters used in the OSCORE group.

In this case, the joining node MUST provide a different compatible authentication credential and public key included thereof to the Group Manager in the 'client_cred' parameter of the Joining Request (see Section 6.1). Then, the Group Manager performs consistency checks on this latest provided authentication credential as per Section 6.2 and, in case of success, considers it as the authentication credential associated with the joining node in the OSCORE group.

- * The joining node and the Group Manager use a symmetric proof-of-possession key to establish a secure communication association. In this case, upon performing a joining process with that Group Manager for the first time, the joining node specifies its own authentication credential in the 'client_cred' parameter of the Joining Request (see Section 6.1).

5. Authorization to Join a Group

This section builds on Section 3 of [I-D.ietf-ace-key-groupcomm] and is organized as follows.

First, Section 5.1 and Section 5.2 describe how the joining node interacts with the AS, in order to be authorized to join an OSCORE group under a given Group Manager and to obtain an Access Token. Then, Section 5.3 describes how the joining node transfers the obtained Access Token to the Group Manager. The following considers a joining node that intends to contact the Group Manager for the first time.

Note that what is defined in Section 3 of [I-D.ietf-ace-key-groupcomm] applies, and only additions or modifications to that specification are defined in this document.

5.1. Authorization Request

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

- * If the 'scope' parameter is present:

- The value of the CBOR byte string encodes a CBOR array, whose format MUST follow the data model AIF-OSCORE-GROUPCOMM defined in Section 3. In particular, for each OSCORE group to join:
 - o The group name is encoded as a CBOR text string.
 - o The set of requested roles is expressed as a single CBOR unsigned integer. This is computed as defined in Section 3, from the numerical abbreviations of each requested role defined in the "Group OSCORE Roles" registry, for which this document defines the entries in Figure 1 (REQ1).

5.2. Authorization Response

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

- * The AS MUST include the 'expires_in' parameter. Other means for the AS to specify the lifetime of Access Tokens are out of the scope of this document.
- * The AS MUST include the 'scope' parameter, when the value included in the Access Token differs from the one specified by the joining node in the Authorization Request. In such a case, the second element of each scope entry MUST be present, and specifies the set of roles that the joining node is actually authorized to take in the OSCORE group for that scope entry, encoded as specified in Section 5.1.

Furthermore, if the AS uses the extended format of scope defined in Section 7 of [I-D.ietf-ace-key-groupcomm] for the 'scope' claim of the Access Token, the first element of the CBOR sequence [RFC8742] MUST be the CBOR integer with value SEM_ID_TBD, defined in Section 16.12 of this document (REQ28). This indicates that the second element of the CBOR sequence, as conveying the actual access control information, follows the scope semantics defined for this application profile in Section 3 of this document.

5.3. Token Transferring

The exchange of Token Transfer Request and Token Transfer Response is defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm]. In addition to that, the following applies.

- * The Token Transfer Request MAY additionally contain the following parameters, which, if included, MUST have the corresponding values defined below (OPT2):

- 'ecdh_info' defined in Section 5.3.1 of this document, with value the CBOR simple value "null" (0xf6) to request information about the ECDH algorithm, the ECDH algorithm parameters, the ECDH key parameters and the exact format of authentication credentials used in the groups that the Client has been authorized to join. This is relevant in case the joining node supports the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm].
- 'kdc_dh_creds' defined in Section 5.3.2 of this document, with value the CBOR simple value "null" (0xf6) to request the Diffie-Hellman authentication credentials of the Group Manager for the groups that the Client has been authorized to join. That is, each of such authentication credentials includes a Diffie-Hellman public key of the Group Manager. This is relevant in case the joining node supports the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

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

- * The 'kdcchallenge' parameter contains a dedicated nonce N_S generated by the Group Manager. For the N_S value, it is RECOMMENDED to use a 8-byte long random nonce. The joining node can use this nonce in order to prove the possession of its own private key, upon joining the group (see Section 6.1).

The 'kdcchallenge' parameter MAY be omitted from the Token Transfer Response, if the 'scope' of the Access Token specifies only the role "monitor" or only the role "verifier" or only the two roles combined, for each and every of the specified groups.

- * If the 'sign_info' parameter is present in the response, the following applies for each element 'sign_info_entry'.
 - 'id' MUST NOT refer to OSCORE groups that are pairwise-only groups.
 - 'sign_alg' takes value from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
 - 'sign_parameters' is a CBOR array. Its format and value are the same of the COSE capabilities array for the algorithm indicated in 'sign_alg', as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [COSE.Algorithms] (REQ4).

- 'sign_key_parameters' is a CBOR array. Its format and value are the same of the COSE capabilities array for the COSE key type of the keys used with the algorithm indicated in 'sign_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types] (REQ5).
- 'pub_key_enc' takes value from the "Label" column of the "COSE Header Parameters" registry [COSE.Header.Parameters] (REQ6). Consistently with Section 2.3 of [I-D.ietf-core-oscore-groupcomm], acceptable values denote a format of authentication credential that MUST explicitly provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

At the time of writing this specification, acceptable formats of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further formats may be available in the future, and would be acceptable to use as long as they comply with the criteria defined above.

[As to CWTs and CCSs, the COSE Header Parameters 'kcwt' and 'kccs' are under pending registration requested by draft-ietf-lake-edhoc.]

[As to C509 certificates, the COSE Header Parameters 'c5b' and 'c5c' are under pending registration requested by draft-ietf-cose-cbor-encoded-cert.]

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

- * If 'ecdh_info' is included in the Token Transfer Request, the Group Manager SHOULD include the 'ecdh_info' parameter in the Token Transfer Response, as per the format defined in Section 5.3.1. Note that the field 'id' of each 'ecdh_info_entry' specifies the name, or array of group names, for which that 'ecdh_info_entry' applies to.

As an exception, the KDC MAY omit the 'ecdh_info' parameter in the Token Transfer Response even if 'ecdh_info' is included in the Token Transfer Request, in case all the groups that the Client is authorized to join are signature-only groups.

- * If 'kdc_dh_creds' is included in the Token Transfer Request and any of the groups that the Client has been authorized to join is a pairwise-only group, then the Group Manager MUST include the 'kdc_dh_creds' parameter in the Token Transfer Response, as per the format defined in Section 5.3.2. Otherwise, if 'kdc_dh_creds' is included in the Token Transfer Request, the Group Manager MAY include the 'kdc_dh_creds' parameter in the Token Transfer Response. Note that the field 'id' specifies the group name, or array of group names, for which the corresponding 'kdc_dh_creds' applies to.

Note that, other than through the above parameters as defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm], the joining node may have obtained such information by alternative means. For example, information conveyed in the 'sign_info' and 'ecdh_info' parameters may have been pre-configured, or the joining node MAY early retrieve it by using the approach described in [I-D.tiloca-core-oscore-discovery], to discover the OSCORE group and the link to the associated group-membership resource at the Group Manager (OPT3).

5.3.1. 'ecdh_info' Parameter

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

This parameter allows the Client and the RS to exchange information about an ECDH algorithm as well as about the authentication credentials and public keys to accordingly use for deriving Diffie-Hellman secrets. Its exact semantics and content are application specific.

In this application profile, this parameter is used to exchange information about the ECDH algorithm as well as about the authentication credentials and public keys to be used with it, in the groups indicated by the transferred Access Token as per its 'scope' claim (see Section 3.2 of [I-D.ietf-ace-key-groupcomm]).

When used in the Token Transfer Request sent to the Group Manager, the 'ecdh_info' parameter has value the CBOR simple value "null" (0xf6). This is done to ask for information about the ECDH algorithm

as well as about the authentication credentials and public keys to be used to compute static-static Diffie-Hellman shared secrets [NIST-800-56A], in the OSCORE groups that the Client has been authorized to join and that use the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

When used in the following Token Transfer Response from the Group Manager, the 'ecdh_info' parameter is a CBOR array of one or more elements. The number of elements is at most the number of OSCORE groups that the Client has been authorized to join.

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

- * The first element 'id' is the group name of the OSCORE group or an array of group names for the OSCORE groups for which the specified information applies. In particular 'id' MUST NOT refer to OSCORE groups that are signature-only groups.
- * The second element 'ecdh_alg' is a CBOR integer or a CBOR text string indicating the ECDH algorithm used in the OSCORE group identified by 'gname'. Values are taken from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
- * The third element 'ecdh_parameters' is a CBOR array indicating the parameters of the ECDH algorithm used in the OSCORE group identified by 'gname'. Its format and value are the same of the COSE capabilities array for the algorithm indicated in 'ecdh_alg', as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [COSE.Algorithms].
- * The fourth element 'ecdh_key_parameters' is a CBOR array indicating the parameters of the keys used with the ECDH algorithm in the OSCORE group identified by 'gname'. Its content depends on the value of 'ecdh_alg'. In particular, its format and value are the same of the COSE capabilities array for the COSE key type of the keys used with the algorithm indicated in 'ecdh_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types].
- * The fifth element 'cred_fmt' is a CBOR integer indicating the format of authentication credentials used in the OSCORE group identified by 'gname'. It takes value from the "Label" column of the "COSE Header Parameters" registry [COSE.Header.Parameters] (REQ6). Acceptable values denote a format that MUST provide the

public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable). The same considerations and guidelines for the 'pub_key_enc' element of 'sign_info' apply (see Section 5.3).

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

```
ecdh_info = ecdh_info_req / ecdh_info_resp

ecdh_info_req = null                                ; in the Token Transfer
                                                       ; Request to the
                                                       ; Group Manager

ecdh_info_res = [ + ecdh_info_entry ] ; in the Token Transfer
                                           ; Response from the
                                           ; Group Manager

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

gname = tstr
```

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

5.3.2. 'kdc_dh_creds' Parameter

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

This parameter allows the Client to request and retrieve the Diffie-Hellman authentication credentials of the RS, i.e., authentication credentials including a Diffie-Hellman public key of the RS.

In this application profile, this parameter is used to request and retrieve from the Group Manager its Diffie-Hellman authentication credentials to use, in the OSCORE groups that the Client has been authorized to join. The Group Manager has specifically a Diffie-Hellman authentication credential in an OSCORE group, and thus a Diffie-Hellman public key in that group, if and only if the group is a pairwise-only group. In this case, the early retrieval of the Group Manager's authentication credential is necessary in order for the joining node to prove the possession of its own private key, upon joining the group (see Section 6.1).

When used in the Token Transfer Request sent to the Group Manager, the 'kdc_dh_creds' parameter has value the CBOR simple value "null" (0xf6). This is done to ask for the Diffie-Hellman authentication credentials that the Group Manager uses in the OSCORE groups that the Client has been authorized to join.

When used in the following Token Transfer Response from the Group Manager, the 'kdc_dh_creds' parameter is a CBOR array of one or more elements. The number of elements is at most the number of OSCORE groups that the Client has been authorized to join.

Each element 'kdc_dh_creds_entry' contains information about the Group Manager's Diffie-Hellman authentication credentials, for one or more OSCORE groups that are pairwise-only groups and that the Client has been authorized to join. Each element is formatted as follows.

- * The first element 'id' is the group name of the OSCORE group or an array of group names for the OSCORE groups for which the specified information applies. In particular 'id' MUST refer exclusively to OSCORE groups that are pairwise-only groups.
- * The second element 'cred_fmt' is a CBOR integer indicating the format of authentication credentials used in the OSCORE group identified by 'gname'. It takes value from the "Label" column of the "COSE Header Parameters" registry [COSE.Header.Parameters] (REQ6). Acceptable values denote a format that MUST explicitly provide the public key as well as comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable). The same considerations and guidelines for the 'pub_key_enc' element of 'sign_info' apply (see Section 5.3).
- * The third element 'cred' is a CBOR byte string, which encodes the Group Manager's Diffie-Hellman authentication credential in its original binary representation made available to other endpoints in the group. In particular, the original binary representation complies with the format specified by the 'cred_fmt' element.

Note that the authentication credential provides the comprehensive set of information related to its public key algorithm, i.e., the ECDH algorithm used in the OSCORE group as pairwise key agreement algorithm.

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

```
kdc_dh_creds = kdc_dh_creds_req / kdc_dh_creds_resp
```

```
kdc_dh_creds_req = null                                     ; in the Token Transfer
                                                         ; Request to the
                                                         ; Group Manager
```

```
kdc_dh_creds_res = [ + kdc_dh_creds_entry ] ; in the Token Transfer
                                                         ; Response from the
                                                         ; Group Manager
```

```
kdc_dh_creds_entry =
[
  id : gname / [ + gname ],
  cred_fmt = int,
  cred = bstr
]
```

```
gname = tstr
```

6. Group Joining

This section describes the interactions between the joining node and the Group Manager to join an OSCORE group. The message exchange between the joining node and the Group Manager consists of the messages defined in Section 4.3.1.1 of [I-D.ietf-ace-key-groupcomm]. Note that what is defined in [I-D.ietf-ace-key-groupcomm] applies, and only additions or modifications to that specification are defined in this document.

6.1. Send the Joining Request

The joining node requests to join the OSCORE group by sending a Joining Request message to the related group-membership resource at the Group Manager, as per Section 4.3.1.1 of [I-D.ietf-ace-key-groupcomm]. Additionally to what is defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], the following applies.

- * The 'scope' parameter MUST be included. Its value encodes one scope entry with the format defined in Section 3, indicating the group name and the role(s) that the joining node wants to take in the group.
- * The 'get_pub_keys' parameter is present only if the joining node wants to retrieve the authentication credentials of the group members from the Group Manager during the joining process (see Section 4). Otherwise, this parameter MUST NOT be present.

If this parameter is present and its value is not the CBOR simple value "null" (0xf6), each element of the inner CBOR array 'role_filter' is encoded as a CBOR unsigned integer, with the same value of a permission set ("Tperm") indicating that role or combination of roles in a scope entry, as defined in Section 3.

- * 'cnonce' contains a dedicated nonce N_C generated by the joining node. For the N_C value, it is RECOMMENDED to use a 8-byte long random nonce.
- * The proof-of-possession (PoP) evidence included in 'client_cred_verify' is computed as defined below (REQ14). In either case, the N_S used to build the PoP input is as defined in Section 6.1.1.
 - If the group is not a pairwise-only group, the PoP evidence MUST be a signature. The joining node computes the signature by using the same private key and signature algorithm it intends to use for signing messages in the OSCORE group.
 - If the group is a pairwise-only group, the PoP evidence MUST be a MAC computed as follows, by using the HKDF Algorithm HKDF SHA-256, which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

MAC = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

- o salt takes as value the empty byte string.
- o IKM is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using the ECDH algorithm used in the OSCORE group. The joining node uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the Group Manager. For X25519 and X448, the procedure is described in Section 5 of [RFC7748].

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

6.1.1. Value of the N_S Challenge

The value of the N_S challenge is determined as follows.

1. If the joining node has provided the Access Token to the Group Manager by means of a Token Transfer Request to the /authz-info endpoint as in Section 5.3, then N_S takes the same value of the most recent 'kdcchallenge' parameter received by the joining node from the Group Manager. This can be either the one specified in the Token Transfer Response, or the one possibly specified in a 4.00 (Bad Request) error response to a following Joining Request (see Section 6.2).
2. If the provisioning of the Access Token to the Group Manager has relied on the DTLS profile of ACE [I-D.ietf-ace-dtls-authorize] with the Access Token as content of the "psk_identity" field of the ClientKeyExchange message [RFC6347], then N_S is an exporter value computed as defined in Section 7.5 of [RFC8446]. Specifically, N_S is exported from the DTLS session between the joining node and the Group Manager, using an empty 'context_value', 32 bytes as 'key_length', and the exporter label "EXPORTER-ACE-Sign-Challenge-coap-group-oscore-app" defined in Section 16.7 of this document.

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

Section 15.3 provides security considerations on the reuse of the N_S challenge.

6.2. Receive the Joining Request

The Group Manager processes the Joining Request as defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], with the following additions.

The Group Manager verifies the PoP evidence contained in 'client_cred_verify' as follows:

- * As PoP input, the Group Manager uses the value of the 'scope' parameter from the Joining Request as a CBOR byte string, concatenated with N_S encoded as a CBOR byte string, concatenated with N_C encoded as a CBOR byte string. In particular, N_S is determined as described in Section 6.1.1, while N_C is the nonce provided in the 'cnonce' parameter of the Joining Request.
- * As public key of the joining node, the Group Manager uses either the one included in the authentication credential retrieved from the 'client_cred' parameter of the Joining Request, or the one from the already stored authentication credential as acquired from previous interactions with the joining node (see Section 4).
- * If the group is not a pairwise-only group, the PoP evidence is a signature. The Group Manager verifies it by using the public key of the joining node, as well as the signature algorithm used in the OSCORE group and possible corresponding parameters.
- * If the group is a pairwise-only group, the PoP evidence is a MAC. The Group Manager recomputes the MAC through the same process taken by the joining node when preparing the value of the 'client_cred_verify' parameter for the Joining Request (see Section 6.1), with the difference that the Group Manager uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the joining node. The verification succeeds if and only if the recomputed MAC is equal to the MAC conveyed as PoP evidence in the Joining Request.

The Group Manager MUST reply with a 5.03 (Service Unavailable) error response in the following cases:

- * There are currently no OSCORE Sender IDs available to assign in the OSCORE group and, at the same time, the joining node is not going to join the group exclusively as monitor. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 4 ("No available node identifiers").
- * The OSCORE group that the joining node has been trying to join is currently inactive (see Section 8.1). The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 9 ("Group currently not active").

The Group Manager MUST reply with a 4.00 (Bad Request) error response in the following cases:

- * The 'client_cred' parameter is present in the Joining Request and its value is not an eligible authentication credential (e.g., it is not of the format accepted in the group).
- * The 'client_cred' parameter is not present in the Joining Request while the joining node is not going to join the group exclusively as monitor, and any of the following conditions holds:
 - The Group Manager does not store an eligible authentication credential (e.g., of the format accepted in the group) for the joining node.
 - The Group Manager stores multiple eligible authentication credentials (e.g., of the format accepted in the group) for the joining node.
- * The 'scope' parameter is not present in the Joining Request, or it is present and specifies any set of roles not included in the following list: "requester", "responder", "monitor", ("requester", "responder"). Future specifications that define a new role for members of OSCORE groups MUST define possible sets of roles (including the new role and existing roles) that are acceptable to specify in the 'scope' parameter of a Joining Request.
- * The Joining Request includes the 'client_cred' parameter but does not include both the 'cnonce' and 'client_cred_verify' parameters.

In order to prevent the acceptance of Ed25519 and Ed448 public keys that cannot be successfully converted to Montgomery coordinates, and thus cannot be used for the derivation of pairwise keys (see Section 2.4.1 of [I-D.ietf-core-oscore-groupcomm]), the Group Manager MAY reply with a 4.00 (Bad Request) error response in case all the following conditions hold:

- * The OSCORE group uses the pairwise mode of Group OSCORE.
- * The OSCORE group uses EdDSA public keys [RFC8032].
- * The authentication credential of the joining node from the 'client_cred' parameter includes a public key which:
 - Is for the elliptic curve Ed25519 and has its Y coordinate equal to -1 or $1 \pmod{p}$, with $p = (2^{255} - 19)$, see Section 4.1 of [RFC7748]; or
 - Is for the elliptic curve Ed448 and has its Y coordinate equal to -1 or $1 \pmod{p}$, with $p = (2^{448} - 2^{224} - 1)$, see Section 4.2 of [RFC7748].

A 4.00 (Bad Request) error response from the Group Manager to the joining node MUST have content format `application/ace-groupcomm+cbor`. The response payload is a CBOR map formatted as follows:

- * If the group uses (also) the group mode of Group OSCORE, the CBOR map MUST contain the `'sign_info'` parameter, whose CBOR label is defined in Section 8 of [I-D.ietf-ace-key-groupcomm]. This parameter has the same format of `'sign_info_res'` defined in Section 3.3.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it includes a single element `'sign_info_entry'` pertaining to the OSCORE group that the joining node has tried to join with the Joining Request.
- * If the group uses (also) the pairwise mode of Group OSCORE, the CBOR map MUST contain the `'ecdh_info'` parameter, whose CBOR label is defined in Section 16.3. This parameter has the same format of `'ecdh_info_res'` defined in Section 5.3.1. In particular, it includes a single element `'ecdh_info_entry'` pertaining to the OSCORE group that the joining node has tried to join with the Joining Request.
- * If the group is a pairwise-only group, the CBOR map MUST contain the `'kdc_dh_creds'` parameter, whose CBOR label is defined in Section 16.3. This parameter has the same format of `'kdc_dh_creds_res'` defined in Section 5.3.2. In particular, it includes a single element `'kdc_dh_creds_entry'` pertaining to the OSCORE group that the joining node has tried to join with the Joining Request.
- * The CBOR map MAY include the `'kdcchallenge'` parameter, whose CBOR label is defined in Section 8 of [I-D.ietf-ace-key-groupcomm]. If present, this parameter is a CBOR byte string, which encodes a newly generated `'kdcchallenge'` value that the Client can use when preparing a Joining Request (see Section 6.1). In such a case the Group Manager MUST store the newly generated value as the `'kdcchallenge'` value associated with the joining node, possibly replacing the currently stored value.

6.2.1. Follow-up to a 4.00 (Bad Request) Error Response

When receiving a 4.00 (Bad Request) error response, the joining node MAY send a new Joining Request to the Group Manager. In such a case:

- * The `'nonce'` parameter MUST include a new dedicated nonce `N_C` generated by the joining node.

- * The 'client_cred' parameter MUST include an authentication credential in the format indicated by the Group Manager. Also, the authentication credential as well as the included public key MUST be compatible with the signature or ECDH algorithm, and possible associated parameters.
- * The 'client_cred_verify' parameter MUST include a PoP evidence computed as described in Section 6.1, by using the private key associated with the authentication credential specified in the current 'client_cred' parameter, with the signature or ECDH algorithm, and possible associated parameters indicated by the Group Manager. If the error response from the Group Manager includes the 'kdcchallenge' parameter, the joining node MUST use its content as new N_S challenge to compute the PoP evidence.

6.3. Send the Joining Response

If the processing of the Joining Request described in Section 6.2 is successful, the Group Manager updates the group membership by registering the joining node NODENAME as a new member of the OSCORE group GROUPNAME, as described in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm].

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

- * The Group Manager selects an available OSCORE Sender ID in the OSCORE group, and exclusively assigns it to the joining node. The Group Manager MUST NOT assign an OSCORE Sender ID to the joining node if this joins the group exclusively with the role of monitor, according to what is specified in the Access Token (see Section 5.2).

Consistently with Section 3.2.1 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MUST assign an OSCORE Sender ID that has not been used in the OSCORE group since the latest time when the current Gid value was assigned to the group.

If the joining node is recognized as a current group member, e.g., through the ongoing secure communication association, the following also applies.

- The Group Manager MUST assign a new OSCORE Sender ID different than the one currently used by the joining node in the OSCORE group.

- The Group Manager MUST add the old, relinquished OSCORE Sender ID of the joining node to the most recent set of stale Sender IDs, in the collection associated with the group (see Section 7.1).
- * The Group Manager stores the association between i) the authentication credential of the joining node; and ii) the Group Identifier (Gid), i.e., the OSCORE ID Context, associated with the OSCORE group together with the OSCORE Sender ID assigned to the joining node in the group. The Group Manager MUST keep this association updated over time.

Then, the Group Manager replies to the joining node, providing the updated security parameters and keying material necessary to participate in the group communication. This success Joining Response is formatted as defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], with the following additions:

- * The 'gkty' parameter identifies a key of type "Group_OSCORE_Input_Material object", defined in Section 16.4 of this document.
- * The 'key' parameter includes what the joining node needs in order to set up the Group OSCORE Security Context as per Section 2 of [I-D.ietf-core-oscore-groupcomm].

This parameter has as value a Group_OSCORE_Input_Material object, which is defined in this document and extends the OSCORE_Input_Material object encoded in CBOR as defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile]. In particular, it contains the additional parameters 'group_senderId', 'cred_fmt', 'sign_enc_alg', 'sign_alg', 'sign_params', 'ecdh_alg' and 'ecdh_params' defined in Section 16.6 of this document.

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

- The 'hkdf' parameter, if present, specifies the HKDF Algorithm used in the OSCORE group. The HKDF Algorithm is specified by the HMAC Algorithm value. This parameter MAY be omitted, if the HKDF Algorithm used in the group is HKDF SHA-256. Otherwise, this parameter MUST be present.
- The 'salt' parameter, if present, has as value the OSCORE Master Salt used in the OSCORE group. This parameter MAY be omitted, if the Master Salt used in the group is the empty byte string. Otherwise, this parameter MUST be present.

- The 'ms' parameter includes the OSCORE Master Secret value used in the OSCORE group. This parameter MUST be present.
- The 'contextId' parameter has as value the Group Identifier (Gid), i.e., the OSCORE ID Context of the OSCORE group. This parameter MUST be present.
- The 'group_senderId' parameter has as value the OSCORE Sender ID assigned to the joining node by the Group Manager, as described above. This parameter MUST be present if and only if the node does not join the OSCORE group exclusively with the role of monitor, according to what is specified in the Access Token (see Section 5.2).
- The 'cred_fmt' parameter specifies the format of authentication credentials used in the OSCORE group. This parameter MUST be present and it takes value from the "Label" column of the "COSE Header Parameters" registry [COSE.Header.Parameters] (REQ6). Consistently with Section 2.3 of [I-D.ietf-core-oscore-groupcomm], acceptable values denote a format that MUST explicitly provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

At the time of writing this specification, acceptable formats of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further formats may be available in the future, and would be acceptable to use as long as they comply with the criteria defined above.

[As to CWTs and CCSs, the COSE Header Parameters 'kcwt' and 'kccs' are under pending registration requested by draft-ietf-lake-edhoc.]

[As to C509 certificates, the COSE Header Parameters 'c5b' and 'c5c' are under pending registration requested by draft-ietf-cose-cbor-encoded-cert.]

The 'key' parameter MUST also include the following parameters, if and only if the OSCORE group is not a pairwise-only group.

- The 'sign_enc_alg' parameter, specifying the Signature Encryption Algorithm used in the OSCORE group to encrypt messages protected with the group mode. This parameter takes values from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
- The 'sign_alg' parameter, specifying the Signature Algorithm used to sign messages in the OSCORE group. This parameter takes values from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
- The 'sign_params' parameter, specifying the parameters of the Signature Algorithm. This parameter is a CBOR array, which includes the following two elements:
 - o 'sign_alg_capab': a CBOR array, with 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].
 - o 'sign_key_type_capab': a CBOR array, with the same format and value of the COSE capabilities array for the COSE key type of the keys used with the Signature Algorithm indicated in 'sign_alg', as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types].

The 'key' parameter MUST also include the following parameters, if and only if the OSCORE group is not a signature-only group.

- The 'alg' parameter, specifying the AEAD Algorithm used in the OSCORE group to encrypt messages protected with the pairwise mode.
- The 'ecdh_alg' parameter, specifying the Pairwise Key Agreement Algorithm used in the OSCORE group. This parameter takes values from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
- The 'ecdh_params' parameter, specifying the parameters of the Pairwise Key Agreement Algorithm. This parameter is a CBOR array, which includes the following two elements:

- o `'ecdh_alg_capab'`: a CBOR array, with the same format and value of the COSE capabilities array for the algorithm indicated in `'ecdh_alg'`, as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [COSE.Algorithms].
- o `'ecdh_key_type_capab'`: a CBOR array, with the same format and value of the COSE capabilities array for the COSE key type of the keys used with the algorithm indicated in `'ecdh_alg'`, as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types].

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

Furthermore, the following applies.

- * The `'exp'` parameter MUST be present.
- * The `'ace-groupcomm-profile'` parameter MUST be present and has value `coap_group_oscore_app` (PROFILE_TBD), which is defined in Section 16.5 of this document.
- * The `'pub_keys'` parameter, if present, includes the authentication credentials requested by the joining node by means of the `'get_pub_keys'` parameter in the Joining Request.

If the joining node has asked for the authentication credentials of all the group members, i.e., `'get_pub_keys'` had value the CBOR simple value `"null"` (0xf6) in the Joining Request, then the Group Manager provides only the authentication credentials of the group members that are relevant to the joining node. That is, in such a case, `'pub_keys'` includes only: i) the authentication credentials of the responders currently in the OSCORE group, in case the joining node is configured (also) as requester; and ii) the authentication credentials of the requesters currently in the OSCORE group, in case the joining node is configured (also) as responder or monitor.

- * The 'peer_identifiers' parameter includes the OSCORE Sender ID of each group member whose authentication credential is specified in the 'pub_keys' parameter. That is, a group member's Sender ID is used as identifier for that group member (REQ25).
- * The 'group_policies' parameter SHOULD be present, and SHOULD include the following elements:
 - "Key Update Check Interval" defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], with default value 3600;
 - "Expiration Delta" defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], with default value 0.
- * The 'kdc_cred' parameter MUST be present, specifying the Group Manager's authentication credential in its original binary representation (REQ8). The Group Manager's authentication credential MUST be in the format used in the OSCORE group. Also, the authentication credential as well as the included public key MUST be compatible with the signature or ECDH algorithm, and possible associated parameters used in the OSCORE group.
- * The 'kdc_nonce' parameter MUST be present, specifying the dedicated nonce N_KDC generated by the Group Manager. For N_KDC, it is RECOMMENDED to use a 8-byte long random nonce.
- * The 'kdc_cred_verify' parameter MUST be present, specifying the proof-of-possession (PoP) evidence computed by the Group Manager. The PoP evidence is computed over the nonce N_KDC, which is specified in the 'kdc_nonce' parameter and taken as PoP input. The PoP evidence is computed as defined below (REQ21).
 - If the group is not a pairwise-only group, the PoP evidence MUST be a signature. The Group Manager computes the signature by using the signature algorithm used in the OSCORE group, as well as its own private key associated with the authentication credential specified in the 'kdc_cred' parameter.
 - If the group is a pairwise-only group, the PoP evidence MUST be a MAC computed as follows, by using the HKDF Algorithm HKDF SHA-256, which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

MAC = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.
 - o salt takes as value the empty byte string.

- o IKM is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using the ECDH algorithm used in the OSCORE group. The Group Manager uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the joining node. For X25519 and X448, the procedure is described in Section 5 of [RFC7748].
 - o info takes as value the PoP input.
 - o L is equal to 8, i.e., the size of the MAC, in bytes.
- * The 'group_rekeying' parameter MAY be omitted, if the Group Manager uses the "Point-to-Point" group rekeying scheme registered in Section 11.14 of [I-D.ietf-ace-key-groupcomm] as rekeying scheme in the OSCORE group (OPT9). Its detailed use for this profile is defined in Section 11 of this document. In any other case, the 'group_rekeying' parameter MUST be included.

As a last action, if the Group Manager reassigns Gid values during the group's lifetime (see Section 3.2.1.1 of [I-D.ietf-core-oscore-groupcomm]), then the Group Manager MUST store the Gid specified in the 'contextId' parameter of the 'key' parameter, as the Birth Gid of the joining node in the joined group (see Section 3 of [I-D.ietf-core-oscore-groupcomm]). This applies also in case the joining node is in fact re-joining the group; in such a case, the newly determined Birth Gid overwrites the one currently stored.

6.4. Receive the Joining Response

Upon receiving the Joining Response, the joining node retrieves the Group Manager's authentication credential from the 'kdc_cred' parameter. The joining node MUST verify the proof-of-possession (PoP) evidence specified in the 'kdc_cred_verify' parameter of the Joining Response as defined below (REQ21).

- * If the group is not a pairwise-only group, the PoP evidence is a signature. The joining node verifies it by using the public key of the Group Manager from the received authentication credential, as well as the signature algorithm used in the OSCORE group and possible corresponding parameters.
- * If the group is a pairwise-only group, the PoP evidence is a MAC. The joining node recomputes the MAC through the same process taken by the Group Manager when computing the value of the 'kdc_cred_verify' parameter (see Section 6.3), with the difference that the joining node uses its own Diffie-Hellman private key and the Diffie-Hellman public key of the Group Manager from the

received authentication credential. The verification succeeds if and only if the recomputed MAC is equal to the MAC conveyed as PoP evidence in the Joining Response.

In case of failed verification of the PoP evidence, the joining node MUST stop processing the Joining Response and MAY send a new Joining Request to the Group Manager (see Section 6.1).

In case of successful verification of the PoP evidence, the joining node uses the information received in the Joining Response to set up the Group OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. If the following parameters were not included in the 'key' parameter of the Joining Response, the joining node considers the default values specified below, consistently with Section 3.2 of [RFC8613].

- * Absent the 'hkdf' parameter, the joining node considers HKDF SHA-256 as HKDF Algorithm to use in the OSCORE group.
- * Absent the 'salt' parameter, the joining node considers the empty byte string as Master Salt to use in the OSCORE group.
- * Absent the 'group_rekeying' parameter, the joining node considers the "Point-to-Point" group rekeying scheme registered in Section 11.14 of [I-D.ietf-ace-key-groupcomm] as the rekeying scheme used in the group (OPT9). Its detailed use for this profile is defined in Section 11 of this document.

In addition, the joining node maintains an association between each authentication credential retrieved from the 'pub_keys' parameter and the role(s) that the corresponding group member has in the OSCORE group.

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

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

If the application requires backward security, the Group Manager MUST generate updated security parameters and group keying material, and provide it to the current group members, upon the new node's joining (see Section 11). In such a case, the joining node is not able to access secure communication in the OSCORE group occurred prior its joining.

7. Overview of the Group Rekeying Process

In a number of cases, the Group Manager has to generate new keying material and distribute it to the group (rekeying), as also discussed in Section 3.2 of [I-D.ietf-core-oscore-groupcomm].

To this end the Group Manager MUST support the Group Rekeying Process described in Section 11 of this document, as an instance of the "Point-to-Point" rekeying scheme defined in Section 6.1 of [I-D.ietf-ace-key-groupcomm] and registered in Section 11.14 of [I-D.ietf-ace-key-groupcomm]. Future documents may define the use of alternative group rekeying schemes for this application profile, together with the corresponding rekeying message formats. The resulting group rekeying process MUST comply with the functional steps defined in Section 3.2 of [I-D.ietf-core-oscore-groupcomm].

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

The data distributed to a group through a rekeying MUST include:

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

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

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

- * A set of stale Sender IDs, which allows each rekeyed node to purge authentication credentials and Recipient Contexts used in the group and associated with those Sender IDs. This in turn allows every group member to rely on stored authentication credentials, in order to confidently assert the group membership of other sender nodes, when receiving protected messages in the group (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]). More details on the maintenance of stale Sender IDs are provided in Section 7.1.

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

The Group Manager MUST rekey the group in the following cases.

- * The application requires backward security - In this case, the group is rekeyed when a node joins the group as a new member. Therefore, a joining node cannot access communications in the group prior its joining.
- * One or more nodes leave the group - That is, the group is rekeyed when one or more current members spontaneously request to leave the group (see Section 9.11), or when the Group Manager forcibly evicts them from the group, e.g., due to expired or revoked authorization (see Section 10). Therefore, a leaving node cannot access communications in the group after its leaving, thus ensuring forward security in the group.

Due to the set of stale Sender IDs distributed through the rekeying, this ensures that a node owning the latest group keying material does not store the authentication credentials of former group members (see Sections 3.2 and 12.1 of [I-D.ietf-core-oscore-groupcomm]).

- * Extension of group lifetime - That is, the group is rekeyed when the expiration time for the group keying material approaches or has passed, if it is appropriate to extend the group operation beyond that.

The Group Manager MAY rekey the group for other reasons, e.g., according to an application-specific rekeying period or scheduling.

7.1. Stale OSCORE Sender IDs

Throughout the lifetime of every group, the Group Manager MUST maintain a collection of stale Sender IDs for that group.

The collection associated with a group MUST include up to $N > 1$ ordered sets of stale OSCORE Sender IDs. It is up to the application to specify the value of N , possibly on a per-group basis.

The N -th set includes the Sender IDs that have become "stale" under the current version V of the group keying material. The $(N - 1)$ -th set refers to the immediately previous version $(V - 1)$ of the group keying material, and so on.

In the following cases, the Group Manager MUST add a new element to the most recent set X , i.e., the set associated with the current version V of the group keying material.

- * When a current group member obtains a new Sender ID, its old Sender ID is added to X . This happens when the Group Manager assigns a new Sender ID upon request from the group member (see Section 9.2), or in case the group member re-joins the group (see Section 6.1 and Section 6.3), thus also obtaining a new Sender ID.
- * When a current group member leaves the group, its current Sender ID is added to X . This happens when a group member requests to leave the group (see Section 9.11) or is forcibly evicted from the group (see Section 10).

The value of N can change throughout the lifetime of the group. If the new value N' is smaller than N , the Group Manager MUST preserve the (up to) N' most recent sets in the collection and MUST delete any possible older set from the collection.

Finally, the Group Manager MUST perform the following actions, when the group is rekeyed and the group shifts to the next version $V' = (V + 1)$ of the group keying material.

1. The Group Manager rekeys the group. This includes also distributing the set of stale Sender IDs X associated with the old group keying material with version V (see Section 7).
2. After completing the group rekeying, the Group Manager creates a new empty set X' associated with the new version V' of the newly established group keying material, i.e., $V' = (V + 1)$.
3. If the current collection of stale Sender IDs has size N , the Group Manager deletes the oldest set in the collection.
4. The Group Manager adds the new set X' to the collection of stale Sender IDs, as the most recent set.

8. Interface at the Group Manager

The Group Manager provides the interface defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm], with the additional sub-resources defined from Section 8.1 to Section 8.3 of this document.

Furthermore, Section 8.4 provides a summary of the CoAP methods admitted to access different resources at the Group Manager, for nodes with different roles in the group or as non members (REQ11).

The GROUPNAME segment of the URI path MUST match with the group name specified in the scope entry of the Access Token scope (i.e., 'gname' in Section 3.1 of [I-D.ietf-ace-key-groupcomm]) (REQ7).

The Resource Type (rt=) Link Target Attribute value "core.osc.gm" is registered in Section 16.11 (REQ10), and can be used to describe group-membership resources and its sub-resources at a Group Manager, e.g., by using a link-format document [RFC6690].

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

8.1. ace-group/GROUPNAME/active

This resource implements a GET handler.

8.1.1. GET Handler

The handler expects a GET request.

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

If all verifications succeed, the handler replies with a 2.05 (Content) response, specifying the current status of the group, i.e., active or inactive. The payload of the response is formatted as defined in Section 9.9.

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

8.2. ace-group/GROUPNAME/verif-data

This resource implements a GET handler.

8.2.1. GET Handler

The handler expects a GET request.

In addition to what is defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm], the Group Manager performs the following checks.

If the requesting Client is a current group member, the Group Manager MUST reply with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 8 ("Operation permitted only to signature verifiers").

If GROUPNAME denotes a pairwise-only group, the Group Manager MUST reply with a 4.00 (Bad Request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 7 ("Signatures not used in the group").

If all verifications succeed, the handler replies with a 2.05 (Content) response, specifying data that allow also an external signature verifier to verify signatures of messages protected with the group mode and sent to the group (see Sections 3.1 and 8.5 of [I-D.ietf-core-oscore-groupcomm]). The response MUST have Content-Format set to application/ace-groupcomm+cbor. The payload of the response is a CBOR map, which is formatted as defined in Section 9.6.

8.3. ace-group/GROUPNAME/stale-sids

This resource implements a FETCH handler.

8.3.1. FETCH Handler

The handler expects a FETCH request, whose payload specifies a version number of the group keying material, encoded as an unsigned CBOR integer.

In addition to what is defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm], the handler verifies that the requesting Client is a current member of the group. If the verification fails, the Group Manager MUST reply with a 4.03

(Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 0 ("Operation permitted only to group members").

If all verifications succeed, the handler replies with a 2.05 (Content) response, specifying data that allow the requesting Client to delete the Recipient Contexts and authentication credentials associated with former members of the group (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]). The payload of the response is formatted as defined in Section 11.3.1.

8.4. Admitted Methods

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

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

CoAP methods: G = GET; F = FETCH; Po = POST; Pu = PUT; D = DELETE

Type1 = Member as Requester and/or Responder

Type2 = Member as Monitor

Type3 = Non-member (authorized to be signature verifier)

(*) = cannot join the group as signature verifier

Type4 = Non-member (not authorized to be signature verifier)

Figure 2: Admitted CoAP Methods on the Group Manager Resources

8.4.1. Signature Verifiers

Just like any candidate group member, a signature verifier provides the Group Manager with an Access Token, as described in Section 5.3. However, unlike candidate group members, it does not join any OSCORE group, i.e., it does not perform the joining process defined in Section 6.

After successfully transferring an Access Token to the Group Manager, a signature verifier is allowed to perform only some operations as non-member of a group, and only for the OSCORE groups specified in the validated Access Token. These are the operations specified in Section 9.3, Section 9.5, Section 9.6 and Section 9.10.

Consistently, in case a node is not a member of the group with group name GROUPNAME and is authorized to be only signature verifier for that group, the Group Manager MUST reply with a 4.03 (Forbidden) error response if that node attempts to access any other endpoint than: /ace-group; ace-group/GROUPNAME/verif-data; /ace-group/GROUPNAME/pub-key; and ace-group/GROUPNAME/kdc-pub-key.

8.5. Operations Supported by Clients

Building on what is defined in Section 4.1.1 of [I-D.ietf-ace-key-groupcomm], and with reference to the resources at the Group Manager newly defined earlier in Section 8 of this document, it is expected that a Client minimally supports also the following set of operations and corresponding interactions with the Group Manager (REQ12).

- * GET request to ace-group/GROUPNAME/active, in order to check the current status of the group.
- * GET request to ace-group/GROUPNAME/verif-data, in order for a signature verifier to retrieve data required to verify signatures of messages protected with the group mode of Group OSCORE and sent to a group (see Sections 3.1 and 8.5 of [I-D.ietf-core-oscore-groupcomm]). Note that this operation is relevant to support only to signature verifiers.
- * FETCH request to ace-group/GROUPNAME/stale-sids, in order to retrieve from the Group Manager the data required to delete some of the stored group members' authentication credentials and associated Recipient Contexts (see Section 8.3.1). These data are provided as an aggregated set of stale Sender IDs, which are used as specified in Section 11.3.

9. Additional Interactions with the Group Manager

This section defines the possible interactions with the Group Manager, in addition to the group joining specified in Section 6.

9.1. Retrieve Updated Keying Material

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

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

9.1.1. Get Group Keying Material

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

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

The Group Manager processes the Key Distribution Request according to Section 4.3.2 of [I-D.ietf-ace-key-groupcomm]. The Key Distribution Response is formatted as defined in Section 4.3.2 of [I-D.ietf-ace-key-groupcomm], with the following additions.

- * The 'key' parameter is formatted as defined in Section 6.3 of this document, with the difference that it does not include the 'group_SenderId' parameter.
- * The 'exp' parameter MUST be present.
- * The 'ace-groupcomm-profile' parameter MUST be present and has value coap_group_oscore_app.

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

9.1.2. Get Group Keying Material and OSCORE Sender ID

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

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

The Group Manager processes the Key Distribution Request according to Section 4.8.1 of [I-D.ietf-ace-key-groupcomm]. The Key Distribution Response is formatted as defined in Section 4.8.1 of [I-D.ietf-ace-key-groupcomm], with the following additions.

- * The 'key' parameter is formatted as defined in Section 6.3 of this document. In particular, if the requesting group member has exclusively the role of monitor, then the 'key' parameter does not include the 'group_SenderId'.

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

- * The 'exp' parameter MUST be present.

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

9.2. Request to Change Individual Keying Material

As discussed in Section 2.5.2 of [I-D.ietf-core-oscore-groupcomm], a group member may at some point exhaust its Sender Sequence Numbers in the OSCORE group.

When this happens, the group member MUST send a Key Renewal Request message to the Group Manager, as per Section 4.8.2.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP PUT request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the Group Manager.

Upon receiving the Key Renewal Request, the Group Manager processes it as defined in Section 4.8.2 of [I-D.ietf-ace-key-groupcomm], with the following additions.

The Group Manager MUST return a 5.03 (Service Unavailable) response in case the OSCORE group identified by GROUPNAME is currently inactive (see Section 8.1). The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 9 ("Group currently not active").

Otherwise, the Group Manager performs one of the following actions.

1. If the requesting group member has exclusively the role of monitor, the Group Manager replies with a 4.03 (Forbidden) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 1 ("Request inconsistent with the current roles").
2. Otherwise, the Group Manager takes one of the following actions.
 - * The Group Manager rekeys the OSCORE group. That is, the Group Manager generates new group keying material for that group (see Section 11), and replies to the group member with a group rekeying message as defined in Section 11, providing the new group keying material. Then, the Group Manager rekeys the rest of the OSCORE group, as discussed in Section 11.

The Group Manager SHOULD perform a group rekeying only if already scheduled to occur shortly, e.g., according to an application-specific rekeying period or scheduling, or as a reaction to a recent change in the group membership. In any other case, the Group Manager SHOULD NOT rekey the OSCORE group when receiving a Key Renewal Request (OPT12).

- * The Group Manager determines and assigns a new OSCORE Sender ID for that group member, and replies with a Key Renewal Response formatted as defined in Section 4.8.2 of [I-D.ietf-ace-key-groupcomm]. In particular, the CBOR Map in the response payload includes a single parameter 'group_SenderId' defined in Section 16.3 of this document, specifying the new Sender ID of the group member encoded as a CBOR byte string.

Consistently with Section 2.5.3.1 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MUST assign a new Sender ID that has not been used in the OSCORE group since the latest time when the current Gid value was assigned to the group.

Furthermore, the Group Manager MUST add the old, relinquished Sender ID of the group member to the most recent set of stale Sender IDs, in the collection associated with the group (see Section 7.1).

The Group Manager MUST return a 5.03 (Service Unavailable) response in case there are currently no Sender IDs available to assign in the OSCORE group. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is

formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 4 ("No available node identifiers").

9.3. Retrieve Authentication Credentials of Group Members

A group member or a signature verifier may need to retrieve the authentication credentials of (other) group members. To this end, the group member or signature verifier sends a Public Key Request message to the Group Manager, as per Sections 4.4.1.1 and 4.4.2.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends the request to the endpoint /ace-group/GROUPNAME/pub-key at the Group Manager.

If the Public Key Request uses the method FETCH, the Public Key Request is formatted as defined in Section 4.4.1 of [I-D.ietf-ace-key-groupcomm]. In particular:

- * Each element (if any) of the inner CBOR array 'role_filter' is formatted as in the inner CBOR array 'role_filter' of the 'get_pub_keys' parameter of the Joining Request when the parameter value is not the CBOR simple value "null" (0xf6) (see Section 6.1).
- * Each element (if any) of the inner CBOR array 'id_filter' is a CBOR byte string, which encodes the OSCORE Sender ID of the group member for which the associated authentication credential is requested (REQ25).

Upon receiving the Public Key Request, the Group Manager processes it as per Section 4.4.1 or Section 4.4.2 of [I-D.ietf-ace-key-groupcomm], depending on the request method being FETCH or GET, respectively. Additionally, if the Public Key Request uses the method FETCH, the Group Manager silently ignores node identifiers included in the 'get_pub_keys' parameter of the request that are not associated with any current group member (REQ26).

The success Public Key Response is formatted as defined in Section 4.4.1 or Section 4.4.2 of [I-D.ietf-ace-key-groupcomm], depending on the request method being FETCH or GET, respectively.

9.4. Upload a New Authentication Credential

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

To this end, the group member sends a Public Key Update Request message to the Group Manager, as per Section 4.9.1.1 of [I-D.ietf-ace-key-groupcomm], with the following addition.

- * The group member computes the proof-of-possession (PoP) evidence included in 'client_cred_verify' in the same way taken when preparing a Joining Request for the OSCORE group in question, as defined in Section 6.1 (REQ14).

In particular, the group member sends a CoAP POST request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME/pub-key at the Group Manager.

Upon receiving the Public Key Update Request, the Group Manager processes it as per Section 4.9.1 of [I-D.ietf-ace-key-groupcomm], with the following additions.

- * The N_S challenge used to build the proof-of-possession input is computed as defined in Section 6.1.1 (REQ15).
- * The Group Manager verifies the PoP challenge included in 'client_cred_verify' in the same way taken when processing a Joining Request for the OSCORE group in question, as defined in Section 6.2 (REQ14).
- * The Group Manager MUST return a 5.03 (Service Unavailable) response in case the OSCORE group identified by GROUPNAME is currently inactive (see Section 8.1). The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 9 ("Group currently not active").
- * If the requesting group member has exclusively the role of monitor, the Group Manager replies with a 4.00 (Bad request) error response. The response MUST have Content-Format set to application/ace-groupcomm+cbor and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 1 ("Request inconsistent with the current roles").
- * If the request is successfully processed, the Group Manager stores the association between i) the new authentication credential of the group member; and ii) the Group Identifier (Gid), i.e., the OSCORE ID Context, associated with the OSCORE group together with the OSCORE Sender ID assigned to the group member in the group. The Group Manager MUST keep this association updated over time.

9.5. Retrieve the Group Manager's Authentication Credential

A group member or a signature verifier may need to retrieve the authentication credential of the Group Manager. To this end, the requesting Client sends a KDC Public Key Request message to the Group Manager.

In particular, it sends a CoAP GET request to the endpoint `/ace-group/GROUPNAME/kdc-pub-key` at the Group Manager defined in Section 4.5.1.1 of [I-D.ietf-ace-key-groupcomm], where GROUPNAME is the name of the OSCORE group.

In addition to what is defined in Section 4.5.1 of [I-D.ietf-ace-key-groupcomm], the Group Manager MUST respond with a 4.00 (Bad Request) error response, if the requesting Client is not a current group member and GROUPNAME denotes a pairwise-only group. The response MUST have Content-Format set to `application/ace-groupcomm+cbor` and is formatted as defined in Section 4.1.2 of [I-D.ietf-ace-key-groupcomm]. The value of the 'error' field MUST be set to 7 ("Signatures not used in the group").

The payload of the 2.05 (Content) KDC Public Key Response is a CBOR map, which is formatted as defined in Section 4.5.1 of [I-D.ietf-ace-key-groupcomm]. In particular, the Group Manager specifies the parameters 'kdc_cred', 'kdc_nonce' and 'kdc_challenge' as defined for the Joining Response in Section 6.3 of this document. This especially applies to the computing of the proof-of-possession (PoP) evidence included in 'kdc_cred_verify' (REQ21).

Upon receiving a 2.05 (Content) KDC Public Key Response, the requesting Client retrieves the Group Manager's authentication credential from the 'kdc_cred' parameter, and proceeds as defined in Section 4.5.1.1 of [I-D.ietf-ace-key-groupcomm]. In particular, the requesting Client verifies the PoP evidence included in 'kdc_cred_verify' by means of the same method used when processing the Joining Response, as defined in Section 6.3 of this document (REQ21).

Note that a signature verifier would not receive a successful response from the Group Manager, in case GROUPNAME denotes a pairwise-only group.

9.6. Retrieve Signature Verification Data

A signature verifier may need to retrieve data required to verify signatures of messages protected with the group mode and sent to a group (see Sections 3.1 and 8.5 of [I-D.ietf-core-oscore-groupcomm]). To this end, the signature verifier sends a Signature Verification Data Request message to the Group Manager.

In particular, it sends a CoAP GET request to the endpoint `/ace-group/GROUPNAME/verif-data` at the Group Manager defined in Section 8.2 of this document, where GROUPNAME is the name of the OSCORE group.

The payload of the 2.05 (Content) Signature Verification Data Response is a CBOR map, which has the format used for the Joining Response message in Section 6.3, with the following differences.

- * From the Joining Response message, only the parameters `'gkty'`, `'key'`, `'num'`, `'exp'` and `'ace-groupcomm-profile'` are present. In particular, the `'key'` parameter includes only the following data.
 - The parameters `'hkdf'`, `'contextId'`, `'cred_fmt'`, `'sign_enc_alg'`, `'sign_alg'`, `'sign_params'`. These parameters MUST be present.
 - The parameters `'alg'` and `'ecdh_alg'`. These parameter MUST NOT be present if the group is a signature-only group. Otherwise, they MUST be present.
- * The parameter `'group_enc_key'` is also included, with CBOR label defined in Section 16.3. This parameter specifies the Group Encryption Key of the OSCORE Group, encoded as a CBOR byte string. The Group Manager derives the Group Encryption Key from the group keying material, as per Section 2.1.6 of [I-D.ietf-core-oscore-groupcomm]. This parameter MUST be present.

In order to verify signatures in the group (see Section 8.5 of [I-D.ietf-core-oscore-groupcomm]), the signature verifier relies on: the data retrieved from the 2.05 (Content) Signature Verification Data Response; the public keys of the group members signing the messages to verify, retrieved from those members' authentication credentials that can be obtained as defined in Section 9.3; and the public key of the Group Manager, retrieved from the Group Manager's authentication credential that can be obtained as defined in Section 9.5.

Figure 3 gives an overview of the exchange described above, while Figure 4 shows an example.

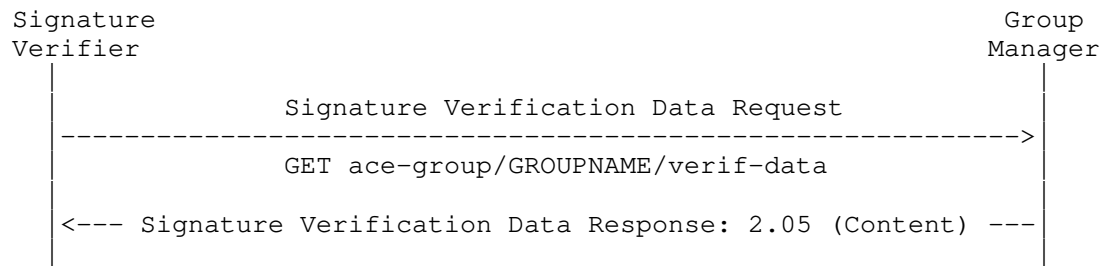


Figure 3: Message Flow of Signature Verification Data Request-Response

Request:

```

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

Response:

```

Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation, with GROUPCOMM_KEY_TBD
        and PROFILE_TBD being CBOR integers, while GROUP_ENC_KEY
        being a CBOR byte string):
{
  "gkty": GROUPCOMM_KEY_TBD,
  "key": {
    'hkdf': 5,                      ; HMAC 256/256
    'contextId': h'37fc',
    'cred_fmt': 33,                  ; x5chain
    'sign_enc_alg': 10,              ; AES-CCM-16-64-128
    'sign_alg': -8,                  ; EdDSA
    'sign_params': [[1], [1, 6]]    ; [[OKP], [OKP, Ed25519]]
  },
  "num": 12,
  "exp": 1609459200,
  "ace_groupcomm_profile": PROFILE_TBD,
  "group_enc_key": GROUP_ENC_KEY
}
  
```

Figure 4: Example of Signature Verification Data Request-Response

9.7. Retrieve the Group Policies

A group member may request the current policies used in the OSCORE group. To this end, the group member sends a Policies Request, as per Section 4.6.1.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/policies at the Group Manager, where GROUPNAME is the name of the OSCORE group.

Upon receiving the Policies Request, the Group Manager processes it as per Section 4.6.1 of [I-D.ietf-ace-key-groupcomm]. The success Policies Response is formatted as defined in Section 4.6.1 of [I-D.ietf-ace-key-groupcomm].

9.8. Retrieve the Keying Material Version

A group member may request the current version of the keying material used in the OSCORE group. To this end, the group member sends a Version Request, as per Section 4.7.1.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/num at the Group Manager, where GROUPNAME is the name of the OSCORE group.

Upon receiving the Version Request, the Group Manager processes it as per Section 4.7.1 of [I-D.ietf-ace-key-groupcomm]. The success Version Response is formatted as defined in Section 4.7.1 of [I-D.ietf-ace-key-groupcomm].

9.9. Retrieve the Group Status

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

In particular, the group member sends a CoAP GET request to the endpoint /ace-group/GROUPNAME/active at the Group Manager defined in Section 8.1 of this document, where GROUPNAME is the name of the OSCORE group.

The payload of the 2.05 (Content) Group Status Response includes the CBOR simple value "true" (0xf5) if the group is currently active, or the CBOR simple value "false" (0xf4) otherwise. The group is considered active if it is set to allow new members to join, and if communication within the group is fine to happen.

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

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

Figure 5 gives an overview of the exchange described above, while Figure 6 shows an example.

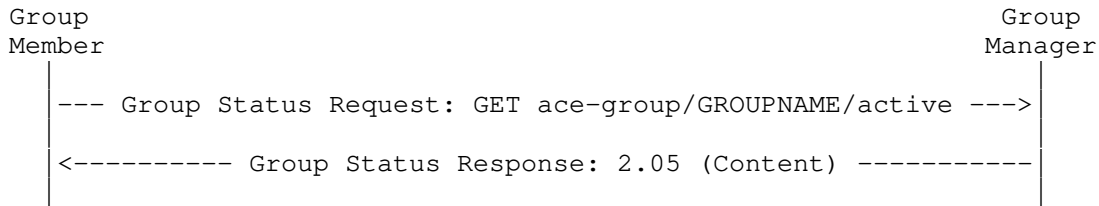


Figure 5: Message Flow of Group Status Request-Response

Request:

```

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

Response:

```

Header: Content (Code=2.05)
Payload (in CBOR diagnostic notation):
  true
  
```

Figure 6: Example of Group Status Request-Response

9.10. Retrieve Group Names

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

- * Before joining a group, a joining node may know only the current Group Identifier (Gid) of that group, but not the group name and the URI to the group-membership resource.
- * As current group member in several groups, the node has missed a previous group rekeying in one of them (see Section 11). Hence, it retains stale keying material and fails to decrypt received messages exchanged in that group.

Such messages do not provide a direct hint to the correct group name, that the node would need in order to retrieve the latest keying material and authentication credentials from the Group Manager (see Section 9.1.1, Section 9.1.2 and Section 9.3). However, such messages may specify the current Gid of the group, as value of the 'kid_context' field of the OSCORE CoAP option (see Section 6.1 of [RFC8613] and Section 4.2 of [I-D.ietf-core-oscore-groupcomm]).

- * As signature verifier, the node also refers to a group name for retrieving the required authentication credentials from the Group Manager (see Section 9.3). As discussed above, intercepted messages do not provide a direct hint to the correct group name, while they may specify the current Gid of the group, as value of the 'kid_context' field of the OSCORE CoAP option. In such a case, upon intercepting a message in the group, the node requires to correctly map the Gid currently used in the group with the invariant group name.

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

In either case, the node only knows the current Gid of the group, as learned from received or intercepted messages exchanged in the group. As detailed below, the node can contact the Group Manager, and request the group name and URI to the group-membership resource corresponding to that Gid. Then, it can use that information to either join the group as a candidate group member, get the latest keying material as a current group member, or retrieve authentication credentials used in the group as a signature verifier. To this end, the node sends a Group Name and URI Retrieval Request, as per Section 4.2.1.1 of [I-D.ietf-ace-key-groupcomm].

In particular, the node sends a CoAP FETCH request to the endpoint /ace-group at the Group Manager formatted as defined in Section 4.2.1 of [I-D.ietf-ace-key-groupcomm]. Each element of the CBOR array 'gid' is a CBOR byte string (REQ13), which encodes the Gid of the group for which the group name and the URI to the group-membership resource are requested.

Upon receiving the Group Name and URI Retrieval Request, the Group Manager processes it as per Section 4.2.1 of [I-D.ietf-ace-key-groupcomm]. The success Group Name and URI Retrieval Response is formatted as defined in Section 4.2.1 of [I-D.ietf-ace-key-groupcomm]. In particular, each element of the CBOR array 'gid' is a CBOR byte string (REQ13), which encodes the Gid of the group for which the group name and the URI to the group-membership resource are provided.

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

Figure 7 gives an overview of the exchanges described above, while Figure 8 shows an example.

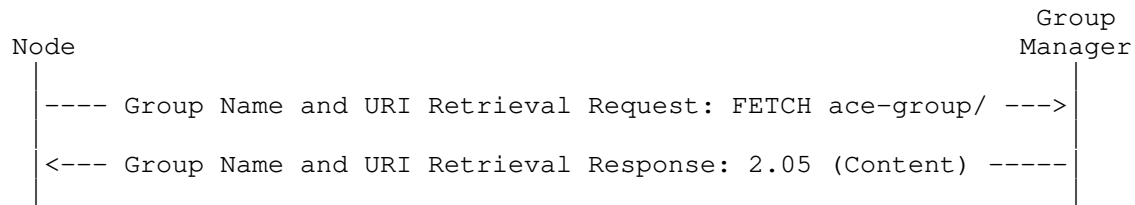


Figure 7: Message Flow of Group Name and URI Retrieval Request-Response

Request:

```
Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
{
  "gid": [h'37fc', h'84bd']
}
```

Response:

```
Header: Content (Code=2.05)
Content-Format: "application/ace-groupcomm+cbor"
Payload (in CBOR diagnostic notation):
{
  "gid": [h'37fc', h'84bd'],
  "gname": ["g1", "g2"],
  "guri": ["ace-group/g1", "ace-group/g2"]
}
```

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

9.11. Leave the Group

A group member may request to leave the OSCORE group. To this end, the group member sends a Group Leaving Request, as per Section 4.8.3.1 of [I-D.ietf-ace-key-groupcomm]. In particular, it sends a CoAP DELETE request to the endpoint /ace-group/GROUPNAME/nodes/NODENAME at the Group Manager.

Upon receiving the Group Leaving Request, the Group Manager processes it as per Section 4.8.3 of [I-D.ietf-ace-key-groupcomm]. Then, the Group Manager performs the follow-up actions defined in Section 10 of this document.

10. Removal of a Group Member

Other than after a spontaneous request to the Group Manager as described in Section 9.11, a node may be forcibly removed from the OSCORE group, e.g., due to expired or revoked authorization.

In either case, if the Group Manager reassigns Gid values during the group's lifetime (see Section 3.2.1.1 of [I-D.ietf-core-oscore-groupcomm]), the Group Manager "forgets" the Birth Gid currently associated with the leaving node in the OSCORE group. This was stored following the Joining Response sent to that node, after its latest (re-)joining of the OSCORE group (see Section 6.3).

If any of the two conditions below holds, the Group Manager MUST inform the leaving node of its eviction as follows. If both conditions hold, the Group Manager MUST inform the leaving node by using only the method corresponding to one of either conditions.

- * If, upon joining the group (see Section 6.1), the leaving node specified a URI in the 'control_uri' parameter defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], the Group Manager sends a DELETE request targeting the URI specified in the 'control_uri' parameter (OPT7).
- * If the leaving node has been observing the associated resource at ace-group/GROUPNAME/nodes/NODENAME, the Group Manager sends an unsolicited 4.04 (Not Found) error response to the leaving node, as specified in Section 4.3.2 of [I-D.ietf-ace-key-groupcomm].

Furthermore, the Group Manager might intend to evict all the current group members from the group at once. In such a case, if the Joining Responses sent by the Group Manager to nodes joining the group (see Section 6.3) specify a URI in the 'control_group_uri' parameter defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm], then the Group Manager MUST additionally send a DELETE request targeting the URI specified in the 'control_group_uri' parameter (OPT10).

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

- * The Group Manager frees the OSCORE Sender ID value of the leaving node. This value MUST NOT become available for possible upcoming joining nodes in the same group, until the group has been rekeyed and assigned a new Group Identifier (Gid).
- * The Group Manager MUST add the relinquished Sender ID of the leaving node to the most recent set of stale Sender IDs, in the collection associated with the group (see Section 7.1).
- * The Group Manager cancels the association between, on one hand, the authentication credential of the leaving node and, on the other hand, the Gid associated with the OSCORE group together with the freed Sender ID value. The Group Manager deletes the

authentication credential of the leaving node, if that authentication credential has no remaining association with any pair (Gid, Sender ID).

Then, the Group Manager MUST generate updated security parameters and group keying material, and provide it to the remaining group members (see Section 11). As a consequence, the leaving node is not able to acquire the new security parameters and group keying material distributed after its leaving.

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

11. Group Rekeying Process

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

As per Section 3.2.1.1 of [I-D.ietf-core-oscore-groupcomm], the Group Manager MAY reassign a Gid to the same group over that group's lifetime, e.g., once the whole space of Gid values has been used for the group in question. If the Group Manager supports reassignment of Gid values and performs it in a group, then the Group Manager additionally takes the following actions.

- * Before rekeying the group, the Group Manager MUST check if the new Gid to be distributed coincides with the Birth Gid of any of the current group members (see Section 6.3).
- * If any of such "elder members" is found in the group, the Group Manager MUST evict them from the group. That is, the Group Manager MUST terminate their membership and MUST rekey the group in such a way that the new keying material is not provided to those evicted elder members. This also includes adding their relinquished Sender IDs to the most recent set of stale Sender IDs, in the collection associated with the group (see Section 7.1), before rekeying the group.

Until a further following group rekeying, the Group Manager MUST store the list of those latest-evicted elder members. If any of those nodes re-joins the group before a further following group rekeying occurs, the Group Manager MUST NOT rekey the group upon their re-joining. When one of those nodes re-joins the group, the Group Manager can rely, e.g., on the ongoing secure communication association to recognize the node as included in the stored list.

Across the rekeying execution, the Group Manager MUST preserve the same unchanged OSCORE Sender IDs for all group members intended to remain in the group. This avoids affecting the retrieval of authentication credentials from the Group Manager and the verification of group messages.

The Group Manager MUST support the "Point-to-Point" group rekeying scheme registered in Section 11.14 of [I-D.ietf-ace-key-groupcomm], as per the detailed use defined in Section 11.1 of this document. Future specifications may define how this application profile can use alternative group rekeying schemes, which MUST comply with the functional steps defined in Section 3.2 of [I-D.ietf-core-oscore-groupcomm]. The Group Manager MUST indicate the use of such an alternative group rekeying scheme to joining nodes, by means of the 'group_rekeying' parameter included in Joining Response messages (see Section 6.3).

It is RECOMMENDED that the Group Manager gets confirmation of successful distribution from the group members, and admits a maximum number of individual retransmissions to non-confirming group members. Once completed the group rekeying process, the Group Manager creates a new empty set X' of stale Sender IDs associated with the version of the newly distributed group keying material. Then, the Group Manager MUST add the set X' to the collection of stale Sender IDs associated with the group (see Section 7.1).

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

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

If the Gid is formatted as described in Appendix C of [I-D.ietf-core-oscore-groupcomm], the Group Prefix can be used as a hint to determine the right Group Manager, as long as no collisions among Group Prefixes are experienced. Otherwise, a group member needs to contact the Group Manager of each group, e.g., by first requesting only the version of the current group keying material (see Section 9.8) and then possibly requesting the current keying material (see Section 9.1.1).

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

Section 11.2 defines the actions performed by a group member upon receiving the new group keying material. Section 11.3 discusses how a group member can realize that it has missed one or more rekeying instances, and the actions it is accordingly required to take.

11.1. Sending Rekeying Messages

When using the "Point-to-Point" group rekeying scheme, the group rekeying messages MUST have Content-Format set to application/ace-groupcomm+cbor and have the same format used for the Joining Response message in Section 6.3, with the following differences. Note that this extends the minimal content of a rekeying message as defined in Section 6 of [I-D.ietf-ace-key-groupcomm] (OPT14).

- * From the Joining Response, only the parameters 'gkty', 'key', 'num', 'exp', and 'ace-groupcomm-profile' are present. In particular, the 'key' parameter includes only the following data.
 - The 'ms' parameter, specifying the new OSCORE Master Secret value. This parameter MUST be present.
 - The 'contextId' parameter, specifying the new Gid to use as OSCORE ID Context value. This parameter MUST be present.
 - The 'salt' value, specifying the new OSCORE Master Salt value. This parameter MAY be present.

- * The parameter 'stale_node_ids' MUST also be included, with CBOR label defined in Section 16.3. This parameter is encoded as a CBOR array, where each element is encoded as a CBOR byte string. The CBOR array has to be intended as a set, i.e., the order of its elements is irrelevant. The parameter is populated as follows.
 - The Group Manager creates an empty CBOR array ARRAY.
 - The Group Manager considers the collection of stale Sender IDs associated with the group (see Section 7.1), and takes the most recent set X, i.e., the set associated with the current version of the group keying material about to be relinquished.
 - For each Sender ID in X, the Group Manager encodes it as a CBOR byte string and adds the result to ARRAY.
 - The parameter 'stale_node_ids' takes ARRAY as value.
- * The parameters 'pub_keys', 'peer_roles' and 'peer_identifiers' SHOULD be present, if the group rekeying is performed due to one or multiple Clients that have requested to join the group. Following the same semantics used in the Joining Response message (see Section 6.3), the three parameters specify the authentication credential, roles in the group and node identifier of each of the Clients that have requested to join the group. The Group Manager MUST NOT include a non-empty subset of these three parameters.

The Group Manager separately sends a group rekeying message formatted as defined above to each group member to be rekeyed.

Each rekeying message MUST be secured with the pairwise secure communication association between the Group Manager and the group member used during the joining process. In particular, each rekeying message can target the 'control_uri' URI path defined in Section 4.3.1 of [I-D.ietf-ace-key-groupcomm] (OPT7), if provided by the intended recipient upon joining the group (see Section 6.1).

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

Group members and the Group Manager SHOULD additionally support alternative distribution approaches that do not require group members to act (also) as servers. A number of such approaches are defined in Section 6 of [I-D.ietf-ace-key-groupcomm]. In particular, a group member may use CoAP Observe [RFC7641] and subscribe for updates to the group-membership resource of the group, at the endpoint /ace-group/GROUPNAME/ of the Group Manager (see Section 6.1 of [I-D.ietf-ace-key-groupcomm]). Alternatively, a full-fledged Pub-Sub model can be considered [I-D.ietf-core-coap-pubsub], where the Group Manager publishes to a rekeying topic hosted at a Broker, while the group members subscribe to such topic (see Section 6.2 of [I-D.ietf-ace-key-groupcomm]).

11.2. Receiving Rekeying Messages

Once received the new group keying material, a group member proceeds as follows. Unless otherwise specified, the following is independent of the specifically used group rekeying scheme.

The group member considers the stale Sender IDs received from the Group Manager. If the "Point-to-Point" group rekeying scheme as detailed in Section 11.1 is used, the stale Sender IDs are specified by the 'stale_node_ids' parameter.

After that, as per Section 3.2 of [I-D.ietf-core-oscore-groupcomm], the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

Then, the following cases can occur, based on the version number V' of the new group keying material distributed through the rekeying process. If the "Point-to-Point" group rekeying scheme as detailed in Section 11.1 is used, this information is specified by the 'num' parameter.

- * The group member has not missed any group rekeying. That is, the old keying material stored by the group member has version number V , while the received new keying material has version number $V' = (V + 1)$. In such a case, the group member simply installs the new keying material and derives the corresponding new Security Context.

- * The group member has missed one or more group rekeying instances. That is, the old keying material stored by the group member has version number V , while the received new keying material has version number $V' > (V + 1)$. In such a case, the group member MUST proceed as defined in Section 11.3.
- * The group member has received keying material not newer than the stored one. That is, the old keying material stored by the group member has version number V , while the received keying material has version number $V' < (V + 1)$. In such a case, the group member MUST ignore the received rekeying messages and MUST NOT install the received keying material.

11.3. Missed Rekeying Instances

A group member can realize to have missed one or more rekeying instances in one of the ways discussed below. In the following, V denotes the version number of the old keying material stored by the group member, while V' denotes the version number of the latest, possibly just distributed, keying material.

- a. The group member has participated to a rekeying process that has distributed new keying material with version number $V' > (V + 1)$, as discussed in Section 11.2.
- b. The group member has obtained the latest keying material from the Group Manager, as a response to a Key Distribution Request (see Section 9.1.1) or to a Joining Request when re-joining the group (see Section 6.1). In particular, V is different than V' specified by the 'num' parameter in the response.
- c. The group member has obtained the authentication credentials of other group members, through a Public Key Request-Response exchange with the Group Manager (see Section 9.3). In particular, V is different than V' specified by the 'num' parameter in the response.
- d. The group member has performed a Version Request-Response exchange with the Group Manager (see Section 9.8). In particular, V is different than V' specified by the 'num' parameter in the response.

In either case, the group member MUST delete the stored keying material with version number V .

If case (a) or case (b) applies, the group member MUST perform the following actions.

1. The group member MUST NOT install the latest keying material yet, in case that was already obtained.
2. The group member sends a Stale Sender IDs Request to the Group Manager (see Section 11.3.1), specifying the version number V as payload of the request.

If the Stale Sender IDs Response from the Group Manager has no payload, the group member MUST remove all the authentication credentials from its list of group members' authentication credentials used in the group, and MUST delete all its Recipient Contexts used in the group.

Otherwise, the group member considers the stale Sender IDs specified in the Stale Sender IDs Response from the Group Manager. Then, the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

3. The group member installs the latest keying material with version number V' and derives the corresponding new Security Context.

If case (c) or case (d) applies, the group member SHOULD perform the following actions.

1. The group member sends a Stale Sender IDs Request to the Group Manager (see Section 11.3.1), specifying the version number V as payload of the request.

If the Stale Sender IDs Response from the Group Manager has no payload, the group member MUST remove all the authentication credentials from its list of group members' authentication credentials used in the group, and MUST delete all its Recipient Contexts used in the group.

Otherwise, the group member considers the stale Sender IDs specified in the Stale Sender IDs Response from the Group Manager. Then, the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

2. The group member obtains the latest keying material with version number V' from the Group Manager. This can happen by sending a Key Distribution Request to the Group Manager (see Section 9.1.1) and Section 9.1.2).
3. The group member installs the latest keying material with version number V' and derives the corresponding new Security Context.

If case (c) or case (d) applies, the group member can alternatively perform the following actions.

1. The group member re-joins the group (see Section 6.1). When doing so, the group member MUST re-join with the same roles it currently has in the group, and MUST request the Group Manager for the authentication credentials of all the current group members. That is, the 'get_pub_keys' parameter of the Joining Request MUST be present and MUST be set to the CBOR simple value "null" (0xf6).
2. When receiving the Joining Response (see Section 6.4 and Section 6.4), the group member retrieves the set Z of authentication credentials specified in the 'pub_keys' parameter.

Then, the group member MUST remove every authentication credential which is not in Z from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group that does not include any of the authentication credentials in Z .

3. The group member installs the latest keying material with version number V' and derives the corresponding new Security Context.

11.3.1. Retrieve Stale Sender IDs

When realizing to have missed one or more group rekeying instances (see Section 11.3), a node needs to retrieve from the Group Manager the data required to delete some of its stored group members' authentication credentials and Recipient Contexts (see Section 8.3.1). These data are provided as an aggregated set of stale Sender IDs, which are used as specified in Section 11.3.

In particular, the node sends a CoAP FETCH request to the endpoint /ace-group/GROUPNAME/stale-sids at the Group Manager defined in Section 8.3 of this document, where GROUPNAME is the name of the OSCORE group.

The payload of the Stale Sender IDs Request MUST include a CBOR unsigned integer. This encodes the version number V of the most recent group keying material stored and installed by the requesting Client, which is older than the latest, possibly just distributed, keying material with version number V' .

The handler MUST reply with a 4.00 (Bad Request) error response, if the request is not formatted correctly. Also, the handler MUST respond with a 4.00 (Bad Request) error response, if the specified version number V is greater or equal than the version number V' associated with the latest keying material in the group, i.e., in case $V \geq V'$.

Otherwise, the handler responds with a 2.05 (Content) Stale Sender IDs Response. The payload of the response is formatted as defined below, where $SKEW = (V' - V + 1)$.

- * The Group Manager considers ITEMS as the current number of sets stored in the collection of stale Sender IDs associated with the group (see Section 7.1).
- * If $SKEW > ITEMS$, the Stale Sender IDs Response MUST NOT have a payload.
- * Otherwise, the payload of the Stale Sender IDs Response MUST include a CBOR array, where each element is encoded as a CBOR byte string. The CBOR array has to be intended as a set, i.e., the order of its elements is irrelevant. The Group Manager populates the CBOR array as follows.
 - The Group Manager creates an empty CBOR array ARRAY and an empty set X.
 - The Group Manager considers the SKEW most recent sets stored in the collection of stale Sender IDs associated with the group. Note that the most recent set is the one associate to the latest version of the group keying material.
 - The Group Manager copies all the Sender IDs from the selected sets into X. When doing so, the Group Manager MUST discard duplicates. That is, the same Sender ID MUST NOT be present more than once in the final content of X.
 - For each Sender ID in X, the Group Manager encodes it as a CBOR byte string and adds the result to ARRAY.

- Finally, ARRAY is specified as payload of the Stale Sender IDs Response. Note that ARRAY might result in the empty CBOR array.

Figure 9 gives an overview of the exchange described above, while Figure 10 shows an example.

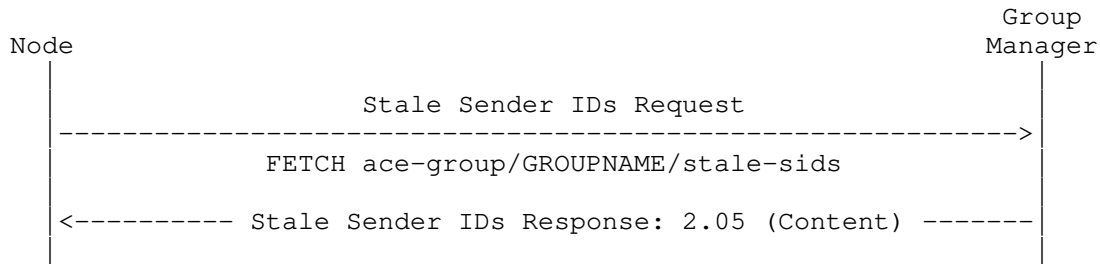


Figure 9: Message Flow of Stale Sender IDs Request-Response

Request:

```

Header: FETCH (Code=0.05)
Uri-Host: "kdc.example.com"
Uri-Path: "ace-group"
Uri-Path: "g1"
Uri-Path: "stale-sids"
Payload (in CBOR diagnostic notation):
  42
  
```

Response:

```

Header: Content (Code=2.05)
Payload (in CBOR diagnostic notation):
  [h'01', h'fc', h'12ab', h'de44', h'ff']
  
```

Figure 10: Example of Stale Sender IDs Request-Response

12. ACE Groupcomm Parameters

In addition to those defined in Section 8 of [I-D.ietf-ace-key-groupcomm], this application profile defines additional parameters used during the second part of the message exchange with the Group Manager, i.e., after the exchange of Token Transfer Request and Response (see Section 5.3). The table below summarizes them and specifies the CBOR key to use instead of the full descriptive name.

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

Name	CBOR Key	CBOR Type	Reference
<code>group_senderId</code>	TBD	<code>bstr</code>	[this document]
<code>ecdh_info</code>	TBD	<code>array</code>	[this document]
<code>kdc_dh_creds</code>	TBD	<code>array</code>	[this document]
<code>group_enc_key</code>	TBD	<code>bstr</code>	[this document]
<code>stale_node_ids</code>	TBD	<code>array</code>	[this document]

Figure 11: ACE Groupcomm Parameters

The Group Manager is expected to support and understand all the parameters above. Instead, a Client is required to support the new parameters defined in this application profile as specified below (REQ29).

- * `'group_senderId'` MUST be supported by a Client that intends to join an OSCORE group with the role of Requester and/or Responder.
- * `'ecdh_info'` MUST be supported by a Client that intends to join a group which uses the pairwise mode of Group OSCORE.
- * `'kdc_dh_creds'` MUST be supported by a Client that intends to join a group which uses the pairwise mode of Group OSCORE and that does not plan to or cannot rely on an early retrieval of the Group Manager's Diffie-Hellman authentication credential.
- * `'group_enc_key'` MUST be supported by a Client that intends to join a group which uses the group mode of Group OSCORE or to be signature verifier for that group.
- * `'stale_node_ids'` MUST be supported.

When the conditional parameters defined in Section 8 of [I-D.ietf-ace-key-groupcomm] are used with this application profile, a Client must, should or may support them as specified below (REQ30).

- * `'client_cred', 'cnonce', 'client_cred_verify'`. A Client that has an own authentication credential to use in a group MUST support these parameters.
- * `'kdcchallenge'`. A Client that has an own authentication credential to use in a group and that provides the Access Token to the Group Manager through a Token Transfer Request (see Section 5.3) MUST support this parameter.
- * `'pub_keys_repo'`. This parameter is not relevant for this application profile, since the Group Manager always acts as repository of the group members' authentication credentials.
- * `'group_policies'`. A Client that is interested in the specific policies used in a group, but that does not know them or cannot become aware of them before joining that group, SHOULD support this parameter.
- * `'peer_roles'`. A Client MUST support this parameter, since in this application profile it is relevant for Clients to know the roles of the group member associated with each authentication credential.
- * `'kdc_nonce', 'kdc_cred' and 'kdc_cred_verify'`. A Client MUST support these parameters, since the Group Manager's authentication credential is required to process messages protected with Group OSCORE (see Section 4.3 of [I-D.ietf-core-oscore-groupcomm]).
- * `'mgt_key_material'`. A Client that supports an advanced rekeying scheme possibly used in the group, such as based on one-to-many rekeying messages sent by the Group Manager (e.g., over IP multicast), MUST support this parameter.
- * `'control_group_uri'`. A Client that supports the hosting of local resources each associated with a group (hence acting as CoAP server) and the reception of one-to-many requests sent to those resources by the Group Manager (e.g., over IP multicast) MUST support this parameter.

13. ACE Groupcomm Error Identifiers

In addition to those defined in Section 9 of [I-D.ietf-ace-key-groupcomm], this application profile defines new values that the Group Manager can include as error identifiers, in the `'error'` field of an error response with Content-Format `application/ace-groupcomm+cbor`.

Value	Description
7	Signatures not used in the group
8	Operation permitted only to signature verifiers
9	Group currently not active

Figure 12: ACE Groupcomm Error Identifiers

A Client supporting the 'error' parameter (see Sections 4.1.2 and 8 of [I-D.ietf-ace-key-groupcomm]) and able to understand the specified error may use that information to determine what actions to take next. If it is included in the error response and supported by the Client, the 'error_description' parameter may provide additional context. In particular, the following guidelines apply.

- * In case of error 7, the Client should stop sending the request in question to the Group Manager. In this application profile, this error is relevant only for a signature verifier, in case it tries to access resources related to a pairwise-only group.
- * In case of error 8, the Client should stop sending the request in question to the Group Manager.
- * In case of error 9, the Client should wait for a certain (pre-configured) amount of time, before trying re-sending its request to the Group Manager.

14. Default Values for Group Configuration Parameters

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

14.1. Common

This section always applies, as related to common configuration parameters.

- * For the HKDF Algorithm 'hkdf', the Group Manager SHOULD use HKDF SHA-256, defined as default in Section 3.2 of [RFC8613]. In the 'hkdf' parameter, this HKDF Algorithm is specified by the HMAC Algorithm HMAC 256/256 (COSE algorithm encoding: 5).

- * For the format 'cred_fmt' used for the authentication credentials in the group, the Group Manager SHOULD use CBOR Web Token (CWT) or CWT Claims Set (CCS) [RFC8392], i.e., the COSE Header Parameter 'kcwt' and 'kccs', respectively.

[These COSE Header Parameters are under pending registration requested by draft-ietf-lake-edhoc.]

- * For 'max_stale_sets', the Group Manager SHOULD consider $N = 3$ as the maximum number of stored sets of stale Sender IDs in the collection associated with the group (see Section 7.1).

14.2. Group Mode

This section applies if the group uses (also) the group mode of Group OSCORE.

- * For the Signature Encryption Algorithm 'sign_enc_alg' used to encrypt messages protected with the group mode, the Group Manager SHOULD use AES-CCM-16-64-128 (COSE algorithm encoding: 10) as default value.

The Group Manager SHOULD use the following default values for the Signature Algorithm 'sign_alg' and related parameters 'sign_params', consistently with the "COSE Algorithms" registry [COSE.Algorithms], the "COSE Key Types" registry [COSE.Key.Types] and the "COSE Elliptic Curves" registry [COSE.Elliptic.Curves].

- * For the Signature Algorithm 'sign_alg' used to sign messages protected with the group mode, the signature algorithm EdDSA [RFC8032].
- * For the parameters 'sign_params' of the Signature Algorithm:
 - The array [[OKP], [OKP, Ed25519]], in case EdDSA is assumed or specified for 'sign_alg'. In particular, this indicates to use the COSE key type OKP and the elliptic curve Ed25519 [RFC8032].
 - The array [[EC2], [EC2, P-256]], in case ES256 [RFC6979] is specified for 'sign_alg'. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-256.
 - The array [[EC2], [EC2, P-384]], in case ES384 [RFC6979] is specified for 'sign_alg'. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-384.

- The array `[[EC2], [EC2, P-521]]`, in case ES512 [RFC6979] is specified for `'sign_alg'`. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-521.
- The array `[[RSA], [RSA]]`, in case PS256, PS384 or PS512 [RFC8017] is specified for `'sign_alg'`. In particular, this indicates to use the COSE key type RSA.

14.3. Pairwise Mode

This section applies if the group uses (also) the pairwise mode of Group OSCORE.

For the AEAD Algorithm `'alg'` used to encrypt messages protected with the pairwise mode, the Group Manager SHOULD use the same default value defined in Section 3.2 of [RFC8613], i.e., AES-CCM-16-64-128 (COSE algorithm encoding: 10).

For the Pairwise Key Agreement Algorithm `'ecdh_alg'` and related parameters `'ecdh_params'`, the Group Manager SHOULD use the following default values, consistently with the "COSE Algorithms" registry [COSE.Algorithms], the "COSE Key Types" registry [COSE.Key.Types] and the "COSE Elliptic Curves" registry [COSE.Elliptic.Curves].

- * For the Pairwise Key Agreement Algorithm `'ecdh_alg'` used to compute static-static Diffie-Hellman shared secrets, the ECDH algorithm ECDH-SS + HKDF-256 specified in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs].
- * For the parameters `'ecdh_params'` of the Pairwise Key Agreement Algorithm:
 - The array `[[OKP], [OKP, X25519]]`, in case EdDSA is assumed or specified for `'sign_alg'`, or in case the group is a pairwise-only group. In particular, this indicates to use the COSE key type OKP and the elliptic curve X25519 [RFC8032].
 - The array `[[EC2], [EC2, P-256]]`, in case ES256 [RFC6979] is specified for `'sign_alg'`. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-256.
 - The array `[[EC2], [EC2, P-384]]`, in case ES384 [RFC6979] is specified for `'sign_alg'`. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-384.
 - The array `[[EC2], [EC2, P-521]]`, in case ES512 [RFC6979] is specified for `'sign_alg'`. In particular, this indicates to use the COSE key type EC2 and the elliptic curve P-521.

15. Security Considerations

Security considerations for this profile are inherited from [I-D.ietf-ace-key-groupcomm], the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the specific transport profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

The following security considerations also apply for this profile.

15.1. Management of OSCORE Groups

This profile leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

- * Management of group keying material (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager is responsible for the renewal and re-distribution of the keying material in the groups of its competence (rekeying).

The Group Manager performs a rekeying when one or more members leave the group, thus preserving forward security and ensuring that the security properties of Group OSCORE are fulfilled. According to the specific application requirements, the Group Manager can also rekey the group upon a new node's joining, in case backward security has also to be preserved.

- * Provisioning and retrieval of authentication credentials (see Section 3 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager acts as repository of authentication credentials of group members, and provides them upon request.
- * Synchronization of sequence numbers (see Section 6.3 of [I-D.ietf-core-oscore-groupcomm]). This concerns how a responder node that has just joined an OSCORE group can synchronize with the sequence number of requesters in the same group.

Before sending the Joining Response, the Group Manager MUST verify that the joining node actually owns the associated private key. To this end, the Group Manager can rely on the proof-of-possession challenge-response defined in Section 6.

Alternatively, when establishing a secure communication association with the Group Manager, the joining node can provide the Group Manager with its own authentication credential, and use the public key included thereof as asymmetric proof-of-possession key. For example, this is the case when the joining node relies on

Section 3.2.2 of [I-D.ietf-ace-dtls-authorize] and authenticates itself during the DTLS handshake with the Group Manager. However, this requires the authentication credential to be in the format used in the OSCORE group, and that both the authentication credential of the joining node and the included public key are compatible with the signature or ECDH algorithm, and possible associated parameters used in the OSCORE group.

A node may have joined multiple OSCORE groups under different non-synchronized Group Managers. Therefore, it can happen that those OSCORE groups have the same Group Identifier (Gid). It follows that, upon receiving a Group OSCORE message addressed to one of those groups, the node would have multiple Security Contexts matching with the Gid in the incoming message. It is up to the application to decide how to handle such collisions of Group Identifiers, e.g., by trying to process the incoming message using one Security Context at the time until the right one is found.

15.2. Size of Nonces as Proof-of-Possession Challenge

With reference to the Joining Request message in Section 6.1, the proof-of-possession (PoP) evidence included in 'client_cred_verify' is computed over an input including also $N_C \parallel N_S$, where \parallel denotes concatenation.

For the N_C challenge, it is RECOMMENDED to use a 8-byte long random nonce. Furthermore, N_C is always conveyed in the 'cnonce' parameter of the Joining Request, which is always sent over the secure communication association between the joining node and the Group Manager.

As defined in Section 6.1.1, the way the N_S value is computed depends on the particular way the joining node provides the Group Manager with the Access Token, as well as on following interactions between the two.

- * If the Access Token has not been provided to the Group Manager by means of a Token Transfer Request to the /authz-info endpoint as in Section 5.3, then N_S is computed as a 32-byte long challenge. For an example, see point (2) of Section 6.1.1.

- * If the Access Token has been provided to the Group Manager by means of a Token Transfer Request to the /authz-info endpoint as in Section 5.3, then N_S takes the most recent value provided to the Client by the Group Manager in the 'kdcchallenge' parameter, as specified in point (1) of Section 6.1.1. This value is provided either in the Token Transfer Response (see Section 5.3), or in a 4.00 (Bad Request) error response to a following Joining Request (see Section 6.2). In either case, it is RECOMMENDED to use a 8-byte long random challenge as value for N_S.

If we consider both N_C and N_S to take 8-byte long values, the following considerations hold.

- * Let us consider both N_C and N_S as taking random values, and the Group Manager to never change the value of the N_S provided to a Client during the lifetime of an Access Token. Then, as per the birthday paradox, the average collision for N_S will happen after 2^{32} new transferred Access Tokens, while the average collision for N_C will happen after 2^{32} new Joining Requests. This amounts to considerably more token provisionings than the expected new joinings of OSCORE groups under a same Group Manager, as well as to considerably more requests to join OSCORE groups from a same Client using a same Access Token under a same Group Manager.
- * Section 7 of [I-D.ietf-ace-oscore-profile] as well Appendix B.2 of [RFC8613] recommend the use of 8-byte random values as well. Unlike in those cases, the values of N_C and N_S considered in this document are not used for as sensitive operations as the derivation of a Security Context, and thus do not have possible implications in the security of AEAD ciphers.

15.3. Reusage of Nonces for Proof-of-Possession Input

As long as the Group Manager preserves the same N_S value currently associated with an Access Token, i.e., the latest value provided to a Client in a 'kdcchallenge' parameter, the Client is able to successfully reuse the same proof-of-possession (PoP) input for multiple Joining Requests to that Group Manager.

In particular, the Client can reuse the same N_C value for every Joining Request to the Group Manager, and combine it with the same unchanged N_S value. This results in reusing the same PoP input for producing the PoP evidence to include in the 'client_cred_verify' parameter of the Joining Requests.

Unless the Group Manager maintains a list of N_C values already used by that Client since the latest update to the N_S value associated with the Access Token, the Group Manager can be forced to falsely

believe that the Client possesses its own private key at that point in time, upon verifying the PoP evidence in the 'client_cred_verify' parameter.

16. IANA Considerations

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

This document has the following actions for IANA.

16.1. OAuth Parameters

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

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

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

16.2. OAuth Parameters CBOR Mappings

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

- * Name: ecdh_info
- * CBOR Key: TBD (range -256 to 255)
- * Value Type: Simple value "null" / Array
- * Reference: [[This document]]

- * Name: kdc_dh_creds
- * CBOR Key: TBD (range -256 to 255)
- * Value Type: Simple value "null" / Array
- * Reference: [[This document]]

16.3. ACE Groupcomm Parameters

IANA is asked to register the following entry to the "ACE Groupcomm Parameters" registry defined in Section 11.7 of [I-D.ietf-ace-key-groupcomm].

- * Name: group_senderId
 - * CBOR Key: TBD
 - * CBOR Type: Byte string
 - * Reference: [[This document]] (Section 9.2)
-
- * Name: ecdh_info
 - * CBOR Key: TBD
 - * CBOR Type: Array
 - * Reference: [[This document]] (Section 6.2)
-
- * Name: kdc_dh_creds
 - * CBOR Key: TBD
 - * CBOR Type: Array
 - * Reference: [[This document]] (Section 6.2)
-
- * Name: group_enc_key
 - * CBOR Key: TBD
 - * CBOR Type: Byte string
 - * Reference: [[This document]] (Section 8.2.1)

- * Name: stale_node_ids
- * CBOR Key: TBD
- * CBOR Type: Array
- * Reference: [[This document]] (Section 11)

16.4. ACE Groupcomm Key Types

IANA is asked to register the following entry to the "ACE Groupcomm Key Types" registry defined in Section 11.8 of [I-D.ietf-ace-key-groupcomm].

- * Name: Group_OSCORE_Input_Material object
- * Key Type Value: GROUPCOMM_KEY_TBD
- * Profile: "coap_group_oscore_app", defined in Section 16.5 of this document.
- * Description: A Group_OSCORE_Input_Material object encoded as described in Section 6.3 of this document.
- * Reference: [[This document]] (Section 6.3)

16.5. ACE Groupcomm Profiles

IANA is asked to register the following entry to the "ACE Groupcomm Profiles" registry defined in Section 11.9 of [I-D.ietf-ace-key-groupcomm].

- * Name: coap_group_oscore_app
- * Description: Application profile to provision keying material for participating in group communication protected with Group OSCORE as per [I-D.ietf-core-oscore-groupcomm].
- * CBOR Value: PROFILE_TBD
- * Reference: [[This document]] (Section 6.3)

16.6. OSCORE Security Context Parameters

IANA is asked to register the following entries in the "OSCORE Security Context Parameters" registry defined in Section 9.4 of [I-D.ietf-ace-oscore-profile].

- * Name: group_SenderId
- * CBOR Label: TBD
- * CBOR Type: Byte string
- * Registry: -
- * Description: OSCORE Sender ID assigned to a member of an OSCORE group
- * Reference: [[This document]] (Section 6.3)

- * Name: cred_fmt
- * CBOR Label: TBD
- * CBOR Type: Integer
- * Registry: COSE Header Parameters
- * Description: Format of authentication credentials to be used in the OSCORE group
- * Reference: [[This document]] (Section 6.3)

- * Name: sign_enc_alg
- * CBOR Label: TBD
- * CBOR Type: Text string / Integer
- * Registry: COSE Algorithms
- * Description: OSCORE Signature Encryption Algorithm Value
- * Reference: [[This document]] (Section 6.3)

- * Name: sign_alg
- * CBOR Label: TBD
- * CBOR Type: Text string / Integer
- * Registry: COSE Algorithms

- * Description: OSCORE Signature Algorithm Value
- * Reference: [[This document]] (Section 6.3)
- * Name: sign_params
- * CBOR Label: TBD
- * CBOR Type: Array
- * Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- * Description: OSCORE Signature Algorithm Parameters
- * Reference: [[This document]] (Section 6.3)
- * Name: ecdh_alg
- * CBOR Label: TBD
- * CBOR Type: Text string / Integer
- * Registry: COSE Algorithms
- * Description: OSCORE Pairwise Key Agreement Algorithm Value
- * Reference: [[This document]] (Section 6.3)
- * Name: ecdh_params
- * CBOR Label: TBD
- * CBOR Type: Array
- * Registry: COSE Algorithms, COSE Key Types, COSE Elliptic Curves
- * Description: OSCORE Pairwise Key Agreement Algorithm Parameters
- * Reference: [[This document]] (Section 6.3)

16.7. TLS Exporter Labels

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

- * Value: EXPORTER-ACE-Sign-Challenge-coap-group-oscore-app
- * DTLS-OK: Y
- * Recommended: N
- * Reference: [[This document]] (Section 6.1.1)

16.8. AIF

For the media-types application/aif+cbor and application/aif+json defined in Section 5.1 of [I-D.ietf-ace-aif], IANA is requested to register the following entries for the two media-type parameters Toid and Tperm, in the respective sub-registry defined in Section 5.2 of [I-D.ietf-ace-aif] within the "MIME Media Type Sub-Parameter" registry group.

- * Name: oscore-gname
- * Description/Specification: OSCORE group name
- * Reference: [[This document]]
- * Name: oscore-gperm
- * Description/Specification: permissions pertaining OSCORE groups
- * Reference: [[This document]]

16.9. CoAP Content-Format

IANA is asked to register the following entries to the "CoAP Content-Formats" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

- * Media Type: application/aif+cbor;Toid="oscore-gname",Tperm="oscore-gperm"
- * Encoding: -
- * ID: TBD
- * Reference: [[This document]]
- * Media Type: application/aif+json;Toid="oscore-gname",Tperm="oscore-gperm"

- * Encoding: -
- * ID: TBD
- * Reference: [[This document]]

16.10. Group OSCORE Roles

This document establishes the IANA "Group OSCORE Roles" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. Expert review guidelines are provided in Section 16.14.

This registry includes the possible roles that nodes can take in an OSCORE group, each in combination with a numeric identifier. These numeric identifiers are used to express authorization information about joining OSCORE groups, as specified in Section 3 of [[This document]].

The columns of this registry are:

- * Name: A value that can be used in documents for easier comprehension, to identify a possible role that nodes can take in an OSCORE group.
- * Value: The numeric identifier for this role. Integer values greater than 65535 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
- * Description: This field contains a brief description of the role.
- * Reference: This contains a pointer to the public specification for the role.

This registry will be initially populated by the values in Figure 1.

The Reference column for all of these entries will be [[This document]].

16.11. CoRE Resource Type

IANA is asked to register the following entry in the "Resource Type (rt=) Link Target Attribute Values" registry within the "Constrained Restful Environments (CoRE) Parameters" registry group.

- * Value: "core.osc.gm"
- * Description: Group-membership resource of an OSCORE Group Manager.

- * Reference: [[This document]]

Client applications can use this resource type to discover a group membership resource at an OSCORE Group Manager, where to send a request for joining the corresponding OSCORE group.

16.12. ACE Scope Semantics

IANA is asked to register the following entry in the "ACE Scope Semantics" registry defined in Section 11.12 of [I-D.ietf-ace-key-groupcomm].

- * Value: SEM_ID_TBD
- * Description: Membership and key management operations at the ACE Group Manager for Group OSCORE.
- * Reference: [[This document]]

16.13. ACE Groupcomm Errors

IANA is asked to register the following entry in the "ACE Groupcomm Errors" registry defined in Section 11.13 of [I-D.ietf-ace-key-groupcomm].

- * Value: 7
- * Description: Signatures not used in the group.
- * Reference: [[This document]]
- * Value: 8
- * Description: Operation permitted only to signature verifiers.
- * Reference: [[This document]]
- * Value: 9
- * Description: Group currently not active.
- * Reference: [[This document]]

16.14. Expert Review Instructions

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

Expert reviewers should take into consideration the following points:

- * Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Experts should inspect the entry for the considered role, to verify the correctness of its description against the role as intended in the specification that defined it. Experts should consider requesting an opinion on the correctness of registered parameters from the Authentication and Authorization for Constrained Environments (ACE) Working Group and the Constrained RESTful Environments (CoRE) Working Group.

Entries that do not meet these objective of clarity and completeness should not be registered.

- * Duplicated registration and point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments.
- * Experts should take into account the expected usage of roles when approving point assignment. Given a 'Value' V as code point, the length of the encoding of $(2^{(V+1)} - 1)$ should be weighed against the usage of the entry, considering the resources and capabilities of devices it will be used on. Additionally, given a 'Value' V as code point, the length of the encoding of $(2^{(V+1)} - 1)$ should be weighed against how many code points resulting in that encoding length are left, and the resources and capabilities of devices it will be used on.
- * Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

17. References

17.1. Normative References

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

[COSE.Elliptic.Curves]
IANA, "COSE Elliptic Curves",
<<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.

[COSE.Header.Parameters]
IANA, "COSE Header Parameters",
<<https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>>.

[COSE.Key.Types]
IANA, "COSE Key Types",
<<https://www.iana.org/assignments/cose/cose.xhtml#key-type>>.

[I-D.ietf-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", Work in Progress, Internet-Draft, draft-ietf-ace-aif-07, 15 March 2022,
<<https://www.ietf.org/archive/id/draft-ietf-ace-aif-07.txt>>.

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-15, 23 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-15.txt>>.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft,

draft-ietf-ace-oauth-authorized-46, 8 November 2021,
<<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authorized-46.txt>>.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson,
"OSCORE Profile of the Authentication and Authorization
for Constrained Environments Framework", Work in Progress,
Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May
2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

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

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

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

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

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

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

[NIST-800-56A]

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

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <<https://www.rfc-editor.org/info/rfc6979>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

- [RFC8447] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <<https://www.rfc-editor.org/info/rfc8447>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [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>>.

17.2. Informative References

- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Höglund, R., Stok, P. V. D., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-05, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-gm-admin-05.txt>>.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>>.
- [I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

- [I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>>.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [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>>.

Appendix A. Profile Requirements

This section lists how this application profile of ACE addresses the requirements defined in Appendix A of [I-D.ietf-ace-key-groupcomm].

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 and Section 5.1.
- * REQ2 - If the AIF format of 'scope' is used, register its specific instance of "Toid" and "Tperm" as Media Type parameters and a corresponding Content-Format, as per the guidelines in [I-D.ietf-ace-aif]: see Section 16.8 and Section 16.9.
- * REQ3 - if used, specify the acceptable values for 'sign_alg': values from the "Value" column of the "COSE Algorithms" registry [COSE.Algorithms].
- * REQ4 - If used, specify the acceptable values for 'sign_parameters': format and values from the COSE algorithm capabilities as specified in the "COSE Algorithms" registry [COSE.Algorithms].
- * REQ5 - If used, specify the acceptable values for 'sign_key_parameters': format and values from the COSE key type capabilities as specified in the "COSE Key Types" registry [COSE.Key.Types].
- * REQ6 - Specify the acceptable formats for authentication credentials and, if used, the acceptable values for 'pub_key_enc': acceptable formats explicitly provide the public key as well as the comprehensive set of information related to the public key algorithm (see Section 5.3 and Section 6.3). Consistent acceptable values for 'pub_key_enc' are taken from the "Label" column of the "COSE Header Parameters" registry [COSE.Header.Parameters].
- * REQ7 - If the value of the GROUPNAME URI path and the group name in the Access Token scope (gname in Section 3.1 of [I-D.ietf-ace-key-groupcomm]) are not required to coincide, specify the mechanism to map the GROUPNAME value in the URI to the group name: not applicable, since a perfect matching is required.

- * REQ8 - Define whether the KDC has an authentication credential and if this has to be provided through the 'kdc_cred' parameter, see Section 4.1 of [I-D.ietf-ace-key-groupcomm]: yes, as required by the Group OSCORE protocol [I-D.ietf-core-oscore-groupcomm], see Section 6.3 of this document.
- * REQ9 - Specify if any part of the KDC interface as defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm] is not supported by the KDC: not applicable.
- * REQ10 - Register a Resource Type for the root url-path, which is used to discover the correct url to access at the KDC (see Section 4.1 of [I-D.ietf-ace-key-groupcomm]): the Resource Type (rt=) Link Target Attribute value "core.osc.gm" is registered in Section 16.11.
- * REQ11 - Define what specific actions (e.g., CoAP methods) are allowed on each resource provided by the KDC interface, depending on whether the Client is a current group member; the roles that a Client is authorized to take as per the obtained access token; and the roles that the Client has as current group member: see Section 8.4.
- * 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 8.5.
- * REQ13 - Specify the encoding of group identifier (see Section 4.2.1 of [I-D.ietf-ace-key-groupcomm]): CBOR byte string (see Section 9.10).
- * REQ14 - Specify the approaches used to compute and verify the PoP evidence to include in 'client_cred_verify', and which of those approaches is used in which case: see Section 6.1 and Section 6.2.
- * REQ15 - Specify how the nonce N_S is generated, if the token is not provided to the KDC through the Token Transfer Request to the authz-info endpoint (e.g., if it is used directly to validate TLS instead): see Section 6.1.1.
- * REQ16 - Define the initial value of the 'num' parameter: the initial value MUST be set to 0 when creating the OSCORE group, e.g., as in [I-D.ietf-ace-oscore-gm-admin].
- * REQ17 - Specify the format of the 'key' parameter: see Section 6.3.

- * REQ18 - Specify acceptable values of the 'gkty' parameter: Group_OSCORE_Input_Material object (see Section 6.3).
- * REQ19 - Specify and register the application profile identifier: coap_group_oscore_app (see Section 16.5).
- * REQ20 - If used, specify the format and content of 'group_policies' and its entries: see Section 6.3.
- * REQ21 - Specify the approaches used to compute and verify the PoP evidence to include in 'kdc_cred_verify', and which of those approaches is used in which case: see Section 6.3, Section 6.4 and Section 9.5.
- * REQ22 - Specify the communication protocol that the members of the group must use: CoAP [RFC7252], also for group communication [I-D.ietf-core-groupcomm-bis].
- * REQ23 - Specify the security protocols that the group members must use to protect their communication: Group OSCORE [I-D.ietf-core-oscore-groupcomm].
- * REQ24 - Specify how the communication is secured between the Client and KDC: by means of any transport profile of ACE [I-D.ietf-ace-oauth-authz] between Client and Group Manager that complies with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].
- * REQ25 - Specify the format of the identifiers of group members: the Sender ID used in the OSCORE group (see Section 6.3 and Section 9.3).
- * REQ26 - Specify policies at the KDC to handle member ids that are not included in 'get_pub_keys': see Section 9.3.
- * REQ27 - Specify the format of newly-generated individual keying material for group members, or of the information to derive it, and corresponding CBOR label: see Section 9.2.
- * REQ28 - Specify and register the identifier of newly defined semantics for binary scopes: see Section 16.12.
- * REQ29 - Categorize newly defined parameters according to the same criteria of Section 8 of [I-D.ietf-ace-key-groupcomm]: see Section 12.

- * REQ30 - Define whether Clients must, should or may support the conditional parameters defined in Section 8 of [I-D.ietf-ace-key-groupcomm], and under which circumstances: see Section 12.

A.2. Optional-to-Address Requirements

- * OPT1 (Optional) - If the textual format of 'scope' is used, specify CBOR values to use for abbreviating the role identifiers in the group: not applicable.
- * OPT2 (Optional) - Specify additional parameters used in the exchange of Token Transfer Request and Response:
 - 'ecdh_info', to negotiate the ECDH algorithm, ECDH algorithm parameters, ECDH key parameters and exact format of authentication credentials used in the group, in case the joining node supports the pairwise mode of Group OSCORE (see Section 5.3).
 - 'kdc_dh_creds', to ask for and retrieve the Group Manager's Diffie-Hellman authentication credentials, in case the joining node supports the pairwise mode of Group OSCORE and the Access Token authorizes to join pairwise-only groups (see Section 5.3).
- * OPT3 (Optional) - Specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' is not used: possible early discovery by using the approach based on the CoRE Resource Directory described in [I-D.tiloca-core-oscore-discovery].
- * OPT4 (Optional) - Specify possible or required payload formats for specific error cases: send a 4.00 (Bad Request) error response to a Joining Request (see Section 6.2).
- * OPT5 (Optional) - Specify additional identifiers of error types, as values of the 'error' field in an error response from the KDC: see Section 16.13.
- * OPT6 (Optional) - Specify the encoding of 'pub_keys_repos' if the default is not used: no.
- * OPT7 (Optional) - Specify the functionalities implemented at the 'control_uri' resource hosted at the Client, including message exchange encoding and other details (see Section 4.3.1 of [I-D.ietf-ace-key-groupcomm]): see Section 10 for the eviction of a group member; see Section 11 for the group rekeying process.

- * OPT8 (Optional) - Specify the behavior of the handler in case of failure to retrieve an authentication credential for the specific node: send a 4.00 (Bad Request) error response to a Joining Request (see Section 6.2).
- * OPT9 (Optional) - Define a default group rekeying scheme, to refer to in case the 'rekeying_scheme' parameter is not included in the Joining Response (see Section 4.3.1.1 of [I-D.ietf-ace-key-groupcomm]): the "Point-to-Point" rekeying scheme registered in Section 11.14 of [I-D.ietf-ace-key-groupcomm], whose detailed use for this profile is defined in Section 11 of this document.
- * OPT10 (Optional) - Specify the functionalities implemented at the 'control_group_uri' resource hosted at the Client, including message exchange encoding and other details (see Section 4.3.1 of [I-D.ietf-ace-key-groupcomm]): see Section 10 for the eviction of multiple group members.
- * OPT11 (Optional) - Specify policies that instruct Clients to retain unsuccessfully decrypted messages and for how long, so that they can be decrypted after getting updated keying material: no.
- * OPT12 (Optional) - Specify for the KDC to perform group rekeying (together or instead of renewing individual keying material) when receiving a Key Renewal Request: the Group Manager SHOULD NOT perform a group rekeying, unless already scheduled to occur shortly (see Section 9.2).
- * OPT13 (Optional) - Specify how the identifier of a group members's authentication credential is included in requests sent to other group members: no.
- * OPT14 (Optional) - Specify additional information to include in rekeying messages for the "Point-to-Point" group rekeying scheme (see Section 6.1 of [I-D.ietf-ace-key-groupcomm]): see Section 11.1.
- * OPT15 (Optional) - Specify if Clients must or should support any of the parameters defined as optional in Section 8 of [I-D.ietf-ace-key-groupcomm]: no.

Appendix B. Extensibility for Future COSE Algorithms

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

To enable the seamless use of such future registered algorithms, this section defines a general, agile format for:

- * Each 'ecdh_info_entry' of the 'ecdh_info' parameter in the Token Transfer Response (see Section 5.3 and Section 5.3.1);
- * The 'sign_params' and 'ecdh_params' parameters within the 'key' parameter (see Section 6.3), as part of the response payloads used in Section 6.3, Section 9.1.1, Section 9.1.2 and Section 11.

Appendix B of [I-D.ietf-ace-key-groupcomm] describes the analogous general format for 'sign_info_entry' of the 'sign_info' parameter in the Token Transfer Response (see Section 5.3 of this document).

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

B.1. Format of 'ecdh_info_entry'

The format of each 'ecdh_info_entry' (see Section 5.3 and Section 5.3.1) is generalized as follows. Given N the number of elements of the 'ecdh_parameters' array, i.e., the number of COSE capabilities of the ECDH algorithm, then:

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

```
ecdh_info_entry =  
[  
  id : gname / [ + gname ],  
  ecdh_alg : int / tstr,  
  ecdh_parameters : [ alg_capab_1 : any,  
                     alg_capab_2 : any,  
                     ...,  
                     alg_capab_N : any ],  
  ecdh_capab_1 : [ any ],  
  ecdh_capab_2 : [ any ],  
  ...,  
  ecdh_capab_N : [ any ],  
  cred_fmt = int / null  
]  
  
gname = tstr
```

Figure 13: 'ecdh_info_entry' with general format

B.2. Format of 'key'

The format of 'key' (see Section 6.3) is generalized as follows.

- * The 'sign_params' array includes N+1 elements, whose exact structure and value depend on the value of the signature algorithm specified in 'sign_alg'.
 - The first element, i.e., 'sign_params'[0], is the array of the N COSE capabilities for the signature algorithm, as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).
 - Each following element 'sign_params'[i], i.e., with index i > 0, is the array of COSE capabilities for the algorithm capability specified in 'sign_params'[0][i-1].

For example, if 'sign_params'[0][0] specifies the key type as capability of the algorithm, then 'sign_params'[1] is the array of COSE capabilities for the COSE key type associated with the signature algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

- * The 'ecdh_params' array includes M+1 elements, whose exact structure and value depend on the value of the ECDH algorithm specified in 'ecdh_alg'.
 - The first element, i.e., 'ecdh_params'[0], is the array of the M COSE capabilities for the ECDH algorithm, as specified for that algorithm in the "Capabilities" column of the "COSE Algorithms" registry [COSE.Algorithms] (see Section 8.1 of [I-D.ietf-cose-rfc8152bis-algs]).
 - Each following element 'ecdh_params'[i], i.e., with index i > 0, is the array of COSE capabilities for the algorithm capability specified in 'ecdh_params'[0][i-1].

For example, if 'ecdh_params'[0][0] specifies the key type as capability of the algorithm, then 'ecdh_params'[1] is the array of COSE capabilities for the COSE key type associated with the ECDH algorithm, as specified for that key type in the "Capabilities" column of the "COSE Key Types" registry [COSE.Key.Types] (see Section 8.2 of [I-D.ietf-cose-rfc8152bis-algs]).

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. Version -13 to -14

- * Major reordering of the document sections.
- * The HKDF Algorithm is specified by the HMAC Algorithm.
- * Group communication does not necessarily use IP multicast.
- * Generalized AIF data model, also for draft-ace-oscore-gm-admin.
- * Clarifications and editorial improvements.

C.2. Version -12 to -13

- * Renamed parameters about authentication credentials.
- * It is optional for the Group Manager to reassign Gids by tracking "Birth Gids".
- * Distinction between authentication credentials and public keys.
- * Updated IANA considerations related to AIF.
- * Updated textual description of registered ACE Scope Semantics value.

C.3. Version -11 to -12

- * Clarified semantics of 'ecdh_info' and 'kdc_dh_creds'.
- * Definition of /ace-group/GROUPNAME/kdc-pub-key moved to draft-ietf-ace-key-groupcomm.
- * ace-group/ accessible also to non-members that are not Verifiers.
- * Clarified what resources are accessible to Verifiers.
- * Revised error handling for the newly defined resources.
- * Revised use of CoAP error codes.
- * Use of "Token Tranfer Request" and "Token Transfer Response".
- * New parameter 'rekeying_scheme'.

- * Categorization of new parameters and inherited conditional parameters.
- * Clarifications on what to do in case of enhanced error responses.
- * Changed UCCS to CCS.
- * Authentication credentials of just joined Clients can be in rekeying messages.
- * Revised names of new IANA registries.
- * Clarified meaning of registered CoRE resource type.
- * Alignment to new requirements from draft-ietf-ace-key-groupcomm.
- * Fixes and editorial improvements.

C.4. Version -10 to -11

- * Removed redundancy of key type capabilities, from 'sign_info', 'ecdh_info' and 'key'.
- * New resource to retrieve the Group Manager's authentication credential.
- * New resource to retrieve material for Signature Verifiers.
- * New parameter 'sign_enc_alg' related to the group mode.
- * 'cred_fmt' takes value from the COSE Header Parameters registry.
- * Improved alignment of the Joining Response payload with the Group OSCORE Security Context parameters.
- * Recycling Group IDs by tracking "Birth GIDs".
- * Error handling in case of non available Sender IDs upon joining.
- * Error handling in case EdDSA public keys with invalid Y coordinate when the pairwise mode of Group OSCORE is supported.
- * Generalized proof-of-possession (PoP) for the joining node's private key; defined Diffie-Hellman based PoP for OSCORE groups using only the pairwise mode.
- * Proof-of-possession of the Group Manager's private key in the Joining Response.

- * Always use 'peer_identifiers' to convey Sender IDs as node identifiers.
- * Stale Sender IDs provided when rekeying the group.
- * New resource for late retrieval of stale Sender IDs.
- * Added examples of message exchanges.
- * Revised default values of group configuration parameters.
- * Fixes to IANA registrations.
- * General format of parameters related to COSE capabilities, supporting future registered COSE algorithms (new Appendix).

C.5. Version -09 to -10

- * Updated non-recycling policy of Sender IDs.
- * Removed policies about Sender Sequence Number synchronization.
- * 'control_path' renamed to 'control_uri'.
- * Format of 'get_pub_keys' aligned with draft-ietf-ace-key-groupcomm.
- * Additional way to inform of group eviction.
- * Registered semantics identifier for extended scope format.
- * Extended error handling, with error type specified in some error responses.
- * Renumbered requirements.

C.6. Version -08 to -09

- * The url-path "ace-group" is used.
- * Added overview of admitted methods on the Group Manager resources.
- * Added exchange of parameters relevant for the pairwise mode of Group OSCORE.
- * The signed value for 'client_cred_verify' includes also the scope.

- * Renamed the key material object as Group_OSCORE_Input_Material object.
- * Replaced 'clientId' with 'group_SenderId'.
- * Added message exchange for Group Names request-response.
- * No reassignment of Sender ID and Gid in the same OSCORE group.
- * Updates on group rekeying contextual with request of new Sender ID.
- * Signature verifiers can also retrieve Group Names and URIs.
- * Removed group policy about supporting Group OSCORE in pairwise mode.
- * Registration of the resource type rt="core.osc.gm".
- * Update list of requirements.
- * Clarifications and editorial revision.

C.7. Version -07 to -08

- * AIF specific data model to express scope entries.
- * A set of roles is checked as valid when processing the Joining Request.
- * Updated format of 'get_pub_keys' in the Joining Request.
- * Payload format and default values of group policies in the Joining Response.
- * Updated payload format of the FETCH request to retrieve authentication credentials.
- * Default values for group configuration parameters.
- * IANA registrations to support the AIF specific data model.

C.8. Version -06 to -07

- * Alignments with draft-ietf-core-oscore-groupcomm.
- * New format of 'sign_info', using the COSE capabilities.

- * New format of Joining Response parameters, using the COSE capabilities.
- * Considerations on group rekeying.
- * Editorial revision.

C.9. Version -05 to -06

- * Added role of external signature verifier.
- * Parameter 'rsnonce' renamed to 'kdcchallenge'.
- * Parameter 'kdcchallenge' may be omitted in some cases.
- * Clarified difference between group name and OSCORE Gid.
- * Removed the role combination ["requester", "monitor"].
- * Admit implicit scope and audience in the Authorization Request.
- * New format for the 'sign_info' parameter.
- * Scope not mandatory to include in the Joining Request.
- * Group policy about supporting Group OSCORE in pairwise mode.
- * Possible individual rekeying of a single requesting node combined with a group rekeying.
- * Security considerations on reuse of signature challenges.
- * Addressing optional requirement OPT12 from draft-ietf-ace-key-groupcomm
- * Editorial improvements.

C.10. Version -04 to -05

- * Nonce N_S also in error responses to the Joining Requests.
- * Supporting single Access Token for multiple groups/topics.
- * Supporting legal requesters/responders using the 'peer_roles' parameter.
- * Registered and used dedicated label for TLS Exporter.

- * Added method for uploading a new authentication credential to the Group Manager.
- * Added resource and method for retrieving the current group status.
- * Fixed inconsistency in retrieving group keying material only.
- * Clarified retrieval of keying material for monitor-only members.
- * Clarification on incrementing version number when rekeying the group.
- * Clarification on what is re-distributed with the group rekeying.
- * Security considerations on the size of the nonces used for the signature challenge.
- * Added CBOR values to abbreviate role identifiers in the group.

C.11. Version -03 to -04

- * New abstract.
- * Moved general content to draft-ietf-ace-key-groupcomm
- * Terminology: node name; node resource.
- * Creation and pointing at node resource.
- * Updated Group Manager API (REST methods and offered services).
- * Size of challenges 'cnonce' and 'rsnonce'.
- * Value of 'rsnonce' for reused or non-traditionally-posted tokens.
- * Removed reference to RFC 7390.
- * New requirements from draft-ietf-ace-key-groupcomm
- * Editorial improvements.

C.12. Version -02 to -03

- * New sections, aligned with the interface of ace-key-groupcomm .
- * Exchange of information on the signature algorithm and related parameters, during the Token POST (Section 4.1).

- * Nonce 'rsnonce' from the Group Manager to the Client (Section 4.1).
- * Client PoP signature in the Key Distribution Request upon joining (Section 4.2).
- * Local actions on the Group Manager, upon a new node's joining (Section 4.2).
- * Local actions on the Group Manager, upon a node's leaving (Section 12).
- * IANA registration in ACE Groupcomm Parameters registry.
- * More fulfilled profile requirements (Appendix A).

C.13. Version -01 to -02

- * Editorial fixes.
- * Changed: "listener" to "responder"; "pure listener" to "monitor".
- * Changed profile name to "coap_group_oscore_app", to reflect it is an application profile.
- * Added the 'type' parameter for all requests to a Join Resource.
- * Added parameters to indicate the encoding of authentication credentials.
- * Challenge-response for proof-of-possession of signature keys (Section 4).
- * Renamed 'key_info' parameter to 'sign_info'; updated its format; extended to include also parameters of the signature key (Section 4.1).
- * Code 4.00 (Bad request), in responses to joining nodes providing an invalid authentication credential (Section 4.3).
- * Clarifications on provisioning and checking of authentication credentials (Sections 4 and 6).
- * Extended discussion on group rekeying and possible different approaches (Section 7).
- * Extended security considerations: proof-of-possession of signature keys; collision of OSCORE Group Identifiers (Section 8).

- * Registered three entries in the IANA registry "Sequence Number Synchronization Method" (Section 9).
- * Registered one public key encoding in the "ACE Public Key Encoding" IANA registry (Section 9).

C.14. Version -00 to -01

- * Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).
- * Added negotiation of signature algorithm/parameters between Client and Group Manager (Section 4).
- * Updated format of the Key Distribution Response as a whole (Section 4.3).
- * Added parameter 'cs_params' in the 'key' parameter of the Key Distribution Response (Section 4.3).
- * New IANA registrations in the "ACE Authorization Server Request Creation Hints" registry, "ACE Groupcomm Key" registry, "OSCORE Security Context Parameters" registry and "ACE Groupcomm Profile" registry (Section 9).

Acknowledgments

The authors sincerely thank Christian Amsuess, Santiago Aragon, Stefan Beck, Carsten Bormann, Martin Gunnarsson, Rikard Hoeglund, Watson Ladd, Daniel Migault, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok for their comments and feedback.

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

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-164 29 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
45127 Essen
Germany
Email: ji-ye.park@uni-due.de

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 12 May 2022

L. Seitz
Combitech
G. Selander
Ericsson
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
Arm Ltd.
8 November 2021

Authentication and Authorization for Constrained Environments (ACE)
using the OAuth 2.0 Framework (ACE-OAuth)
draft-ietf-ace-oauth-authz-46

Abstract

This specification defines a framework for authentication and authorization in Internet of Things (IoT) environments called ACE-OAuth. The framework is based on a set of building blocks including OAuth 2.0 and the Constrained Application Protocol (CoAP), thus transforming a well-known and widely used authorization solution into a form suitable for IoT devices. Existing specifications are used where possible, but extensions are added and profiles are defined to better serve the IoT use cases.

Status of This Memo

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

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

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

This Internet-Draft will expire on 12 May 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
2. Terminology	5
3. Overview	6
3.1. OAuth 2.0	7
3.2. CoAP	10
4. Protocol Interactions	11
5. Framework	14
5.1. Discovering Authorization Servers	16
5.2. Unauthorized Resource Request Message	16
5.3. AS Request Creation Hints	17
5.3.1. The Client-Nonce Parameter	19
5.4. Authorization Grants	20
5.5. Client Credentials	20
5.6. AS Authentication	21
5.7. The Authorization Endpoint	21
5.8. The Token Endpoint	21
5.8.1. Client-to-AS Request	22
5.8.2. AS-to-Client Response	25
5.8.3. Error Response	27
5.8.4. Request and Response Parameters	28
5.8.4.1. Grant Type	28
5.8.4.2. Token Type	29
5.8.4.3. Profile	29
5.8.4.4. Client-Nonce	30
5.8.5. Mapping Parameters to CBOR	30
5.9. The Introspection Endpoint	31
5.9.1. Introspection Request	32
5.9.2. Introspection Response	33
5.9.3. Error Response	34
5.9.4. Mapping Introspection Parameters to CBOR	35
5.10. The Access Token	36
5.10.1. The Authorization Information Endpoint	36

5.10.1.1.	Verifying an Access Token	38
5.10.1.2.	Protecting the Authorization Information Endpoint	39
5.10.2.	Client Requests to the RS	40
5.10.3.	Token Expiration	41
5.10.4.	Key Expiration	42
6.	Security Considerations	43
6.1.	Protecting Tokens	43
6.2.	Communication Security	44
6.3.	Long-Term Credentials	44
6.4.	Unprotected AS Request Creation Hints	45
6.5.	Minimal Security Requirements for Communication	45
6.6.	Token Freshness and Expiration	46
6.7.	Combining Profiles	47
6.8.	Unprotected Information	47
6.9.	Identifying Audiences	48
6.10.	Denial of Service Against or with Introspection	49
7.	Privacy Considerations	49
8.	IANA Considerations	50
8.1.	ACE Authorization Server Request Creation Hints	50
8.2.	CoRE Resource Type Registry	51
8.3.	OAuth Extensions Error Registration	51
8.4.	OAuth Error Code CBOR Mappings Registry	52
8.5.	OAuth Grant Type CBOR Mappings	52
8.6.	OAuth Access Token Types	53
8.7.	OAuth Access Token Type CBOR Mappings	53
8.7.1.	Initial Registry Contents	53
8.8.	ACE Profile Registry	54
8.9.	OAuth Parameter Registration	54
8.10.	OAuth Parameters CBOR Mappings Registry	54
8.11.	OAuth Introspection Response Parameter Registration	55
8.12.	OAuth Token Introspection Response CBOR Mappings Registry	56
8.13.	JSON Web Token Claims	56
8.14.	CBOR Web Token Claims	57
8.15.	Media Type Registrations	58
8.16.	CoAP Content-Format Registry	58
8.17.	Expert Review Instructions	59
9.	Acknowledgments	60
10.	References	60
10.1.	Normative References	60
10.2.	Informative References	63
Appendix A.	Design Justification	66
Appendix B.	Roles and Responsibilities	69
Appendix C.	Requirements on Profiles	71
Appendix D.	Assumptions on AS Knowledge about C and RS	72
Appendix E.	Differences to OAuth 2.0	73
Appendix F.	Deployment Examples	73

F.1. Local Token Validation	74
F.2. Introspection Aided Token Validation	78
Authors' Addresses	82

1. Introduction

Authorization is the process for granting approval to an entity to access a generic resource [RFC4949]. The authorization task itself can best be described as granting access to a requesting client, for a resource hosted on a device, the resource server (RS). This exchange is mediated by one or multiple authorization servers (AS). Managing authorization for a large number of devices and users can be a complex task.

While prior work on authorization solutions for the Web and for the mobile environment also applies to the Internet of Things (IoT) environment, many IoT devices are constrained, for example, in terms of processing capabilities, available memory, etc. For such devices the Constrained Application Protocol (CoAP) [RFC7252] can alleviate some resource concerns when used instead of HTTP to implement the communication flows of this specification.

Appendix A gives an overview of the constraints considered in this design, and a more detailed treatment of constraints can be found in [RFC7228]. This design aims to accommodate different IoT deployments and thus a continuous range of device and network capabilities. Taking energy consumption as an example: At one end there are energy-harvesting or battery powered devices which have a tight power budget, on the other end there are mains-powered devices, and all levels in between.

Hence, IoT devices may be very different in terms of available processing and message exchange capabilities and there is a need to support many different authorization use cases [RFC7744].

This specification describes a framework for authentication and authorization in constrained environments (ACE) built on re-use of OAuth 2.0 [RFC6749], thereby extending authorization to Internet of Things devices. This specification contains the necessary building blocks for adjusting OAuth 2.0 to IoT environments.

Profiles of this framework are available in separate specifications, such as [I-D.ietf-ace-dtls-authorize] or [I-D.ietf-ace-oscore-profile]. Such profiles may specify the use of the framework for a specific security protocol and the underlying transports for use in a specific deployment environment to improve interoperability. Implementations may claim conformance with a specific profile, whereby implementations utilizing the same profile

interoperate, while implementations of different profiles are not expected to be interoperable. More powerful devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of constrained devices. Requirements on profiles are described at contextually appropriate places throughout this specification, and also summarized in Appendix C.

2. Terminology

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

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

Since exchanges in this specification are described as RESTful protocol interactions, HTTP [RFC7231] offers useful terminology.

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] such as client (C), resource server (RS), and authorization server (AS).

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as token and introspection at the AS and authz-info at the RS (see Section 5.10.1 for a definition of the authz-info endpoint). The CoAP [RFC7252] definition, which is "An entity participating in the CoAP protocol" is not used in this specification.

The specifications in this document is called the "framework" or "ACE framework". When referring to "profiles of this framework" it refers to additional specifications that define the use of this specification with concrete transport and communication security protocols (e.g., CoAP over DTLS).

The term "Access Information" is used for parameters, other than the access token, provided to the client by the AS to enable it to access the RS (e.g. public key of the RS, profile supported by RS).

The term "Authorization Information" is used to denote all information, including the claims of relevant access tokens, that an RS uses to determine whether an access request should be granted.

3. Overview

This specification defines the ACE framework for authorization in the Internet of Things environment. It consists of a set of building blocks.

The basic block is the OAuth 2.0 [RFC6749] framework, which enjoys widespread deployment. Many IoT devices can support OAuth 2.0 without any additional extensions, but for certain constrained settings additional profiling is needed.

Another building block is the lightweight web transfer protocol CoAP [RFC7252], for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP, which further reduces overhead and message exchanges. While this specification defines extensions for the use of OAuth over CoAP, other underlying protocols are not prohibited from being supported in the future, such as HTTP/2 [RFC7540], Message Queuing Telemetry Transport (MQTT) [MQTT5.0], Bluetooth Low Energy (BLE) [BLE] and QUIC [I-D.ietf-quic-transport]. Note that this document specifies protocol exchanges in terms of RESTful verbs such as GET and POST. Future profiles using protocols that do not support these verbs MUST specify how the corresponding protocol messages are transmitted instead.

A third building block is the Concise Binary Object Representation (CBOR) [RFC8949], for encodings where JSON [RFC8259] is not sufficiently compact. CBOR is a binary encoding designed for small code and message size. Self-contained tokens and protocol message payloads are encoded in CBOR when CoAP is used. When CoAP is not used, the use of CBOR remains RECOMMENDED.

A fourth building block is CBOR Object Signing and Encryption (COSE) [RFC8152], which enables object-level layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC8446]). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which are an extension to the OAuth bearer tokens. The default token format is defined in CBOR Web Token (CWT) [RFC8392]. Application-layer security for CoAP using COSE can be provided with OSCORE [RFC8613].

With the building blocks listed above, solutions satisfying various IoT device and network constraints are possible. A list of constraints is described in detail in [RFC7228] and a description of how the building blocks mentioned above relate to the various constraints can be found in Appendix A.

Luckily, not every IoT device suffers from all constraints. The ACE framework nevertheless takes all these aspects into account and allows several different deployment variants to co-exist, rather than mandating a one-size-fits-all solution. It is important to cover the wide range of possible interworking use cases and the different requirements from a security point of view. Once IoT deployments mature, popular deployment variants will be documented in the form of ACE profiles.

3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain scoped access to a resource with the permission of a resource owner. Authorization information, or references to it, is passed between the nodes using access tokens. These access tokens are issued to clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

A number of OAuth 2.0 terms are used within this specification:

Access Tokens:

Access tokens are credentials needed to access protected resources. An access token is a data structure representing authorization permissions issued by the AS to the client. Access tokens are generated by the AS and consumed by the RS. The access token content is opaque to the client.

Access tokens can have different formats, and various methods of utilization e.g., cryptographic properties) based on the security requirements of the given deployment.

Introspection:

Introspection is a method for a resource server or potentially a client, to query the authorization server for the active state and content of a received access token. This is particularly useful in those cases where the authorization decisions are very dynamic and/or where the received access token itself is an opaque reference rather than a self-contained token. More information about introspection in OAuth 2.0 can be found in [RFC7662].

Refresh Tokens:

Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token expires, or to obtain additional access tokens with identical or narrower scope (such access tokens may have a shorter

lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token (i.e., step (B) in Figure 1).

A refresh token in OAuth 2.0 is a string representing the authorization granted to the client by the resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers. In this framework, refresh tokens are encoded in binary instead of strings, if used.

Proof of Possession Tokens:

A token may be bound to a cryptographic key, which is then used to bind the token to a request authorized by the token. Such tokens are called proof-of-possession tokens (or PoP tokens).

The proof-of-possession security concept used here assumes that the AS acts as a trusted third party that binds keys to tokens. In the case of access tokens, these so called PoP keys are then used by the client to demonstrate the possession of the secret to the RS when accessing the resource. The RS, when receiving an access token, needs to verify that the key used by the client matches the one bound to the access token. When this specification uses the term "access token" it is assumed to be a PoP access token unless specifically stated otherwise.

The key bound to the token (the PoP key) may use either symmetric or asymmetric cryptography. The appropriate choice of the kind of cryptography depends on the constraints of the IoT devices as well as on the security requirements of the use case.

Symmetric PoP key:

The AS generates a random symmetric PoP key. The key is either stored to be returned on introspection calls or included in the token. Either the whole token or only the key MUST be encrypted in the latter case. The PoP key is also returned to client together with the token.

Asymmetric PoP key:

An asymmetric key pair is generated by the client and the public key is sent to the AS (if it does not already have knowledge of the client's public key). Information about the

public key, which is the PoP key in this case, is either stored to be returned on introspection calls or included inside the token and sent back to the client. The resource server consuming the token can identify the public key from the information in the token, which allows the client to use the corresponding private key for the proof of possession.

The token is either a simple reference, or a structured information object (e.g., CWT [RFC8392]) protected by a cryptographic wrapper (e.g., COSE [RFC8152]). The choice of PoP key does not necessarily imply a specific credential type for the integrity protection of the token.

Scopes and Permissions:

In OAuth 2.0, the client specifies the type of permissions it is seeking to obtain (via the scope parameter) in the access token request. In turn, the AS may use the scope response parameter to inform the client of the scope of the access token issued. As the client could be a constrained device as well, this specification defines the use of CBOR encoding, see Section 5, for such requests and responses.

The values of the scope parameter in OAuth 2.0 are expressed as a list of space-delimited, case-sensitive strings, with a semantic that is well-known to the AS and the RS. More details about the concept of scopes is found under Section 3.3 in [RFC6749].

Claims:

Information carried in the access token or returned from introspection, called claims, is in the form of name-value pairs. An access token may, for example, include a claim identifying the AS that issued the token (via the "iss" claim) and what audience the access token is intended for (via the "aud" claim). The audience of an access token can be a specific resource or one or many resource servers. The resource owner policies influence what claims are put into the access token by the authorization server.

While the structure and encoding of the access token varies throughout deployments, a standardized format has been defined with the JSON Web Token (JWT) [RFC7519] where claims are encoded as a JSON object. In [RFC8392] the CBOR Web Token (CWT) has been defined as an equivalent format using CBOR encoding.

The token and introspection Endpoints:

The AS hosts the token endpoint that allows a client to request access tokens. The client makes a POST request to the token endpoint on the AS and receives the access token in the response (if the request was successful).

In some deployments, a token introspection endpoint is provided by the AS, which can be used by the RS and potentially the client, if they need to request additional information regarding a received access token. The requesting entity makes a POST request to the introspection endpoint on the AS and receives information about the access token in the response. (See "Introspection" above.)

3.2. CoAP

CoAP is an application-layer protocol similar to HTTP, but specifically designed for constrained environments. CoAP typically uses datagram-oriented transport, such as UDP, where reordering and loss of packets can occur. A security solution needs to take the latter aspects into account.

While HTTP uses headers and query strings to convey additional information about a request, CoAP encodes such information into header parameters called 'options'.

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [RFC7959]. However, blockwise transfer does not increase the size limits of CoAP options, therefore data encoded in options has to be kept small.

Transport layer security for CoAP can be provided by DTLS or TLS [RFC6347][RFC8446] [I-D.ietf-tls-dtls13]. CoAP defines a number of proxy operations that require transport layer security to be terminated at the proxy. One approach for protecting CoAP communication end-to-end through proxies, and also to support security for CoAP over a different transport in a uniform way, is to provide security at the application layer using an object-based security mechanism such as COSE [RFC8152].

One application of COSE is OSCORE [RFC8613], which provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages. In OSCORE, the CoAP messages are wrapped in COSE objects and sent using CoAP.

In this framework the use of CoAP as replacement for HTTP is RECOMMENDED for use in constrained environments. For communication security this framework does not make an explicit protocol recommendation, since the choice depends on the requirements of the

specific application. DTLS [RFC6347], [I-D.ietf-tls-dtls13] and OSCORE [RFC8613] are mentioned as examples, other protocols fulfilling the requirements from Section 6.5 are also applicable.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the token endpoint and optionally the introspection endpoint. A client obtains an access token, and optionally a refresh token, from an AS using the token endpoint and subsequently presents the access token to an RS to gain access to a protected resource. In most deployments the RS can process the access token locally, however in some cases the RS may present it to the AS via the introspection endpoint to get fresh information. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in Section 3.1.

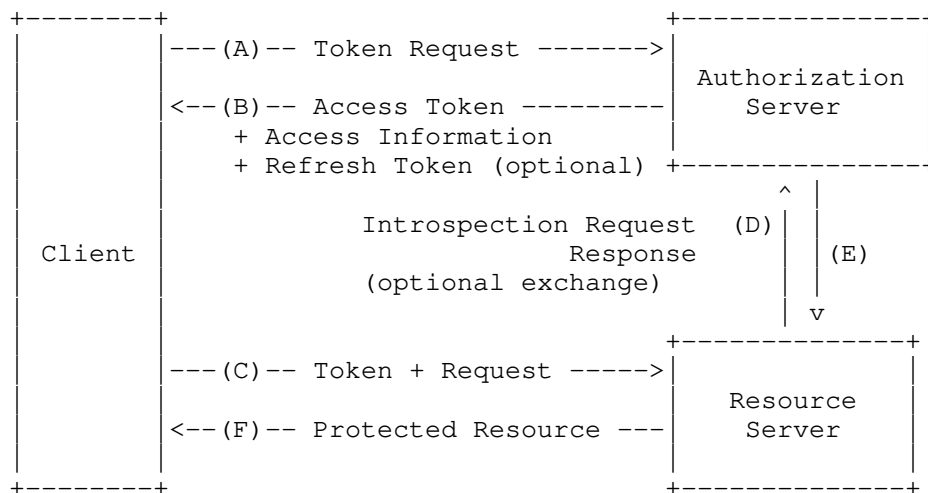


Figure 1: Basic Protocol Flow.

Requesting an Access Token (A):

The client makes an access token request to the token endpoint at the AS. This framework assumes the use of PoP access tokens (see Section 3.1 for a short description) wherein the AS binds a key to an access token. The client may include permissions it seeks to obtain, and information about the credentials it wants to use for proof-of-possession (e.g., symmetric/asymmetric cryptography or a reference to a specific key) of the access token.

Access Token Response (B):

If the request from the client has been successfully verified, authenticated, and authorized, the AS returns an access token and optionally a refresh token. Note that only certain grant types support refresh tokens. The AS can also return additional parameters, referred to as "Access Information". In addition to the response parameters defined by OAuth 2.0 and the PoP access token extension, this framework defines parameters that can be used to inform the client about capabilities of the RS, e.g. the profile the RS supports. More information about these parameters can be found in Section 5.8.4.

Resource Request (C):

The client interacts with the RS to request access to the protected resource and provides the access token. The protocol to use between the client and the RS is not restricted to CoAP. HTTP, HTTP/2 [RFC7540], QUIC [I-D.ietf-quic-transport], MQTT [MQTT5.0], Bluetooth Low Energy [BLE], etc., are also viable candidates.

Depending on the device limitations and the selected protocol, this exchange may be split up into two parts:

- (1) the client sends the access token containing, or referencing, the authorization information to the RS, that will be used for subsequent resource requests by the client, and
- (2) the client makes the resource access request, using the communication security protocol and other Access Information obtained from the AS.

The client and the RS mutually authenticate using the security protocol specified in the profile (see step B) and the keys obtained in the access token or the Access Information. The RS verifies that the token is integrity protected and originated by the AS. It then compares the claims contained in the access token with the resource request. If the RS is online, validation can be handed over to the AS using token introspection (see messages D and E) over HTTP or CoAP.

Token Introspection Request (D):

A resource server may be configured to introspect the access token by including it in a request to the introspection endpoint at that AS. Token introspection over CoAP is defined in Section 5.9 and for HTTP in [RFC7662].

Note that token introspection is an optional step and can be omitted if the token is self-contained and the resource server is prepared to perform the token validation on its own.

Token Introspection Response (E):

The AS validates the token and returns the most recent parameters, such as scope, audience, validity etc. associated with it back to the RS. The RS then uses the received parameters to process the request to either accept or to deny it.

Protected Resource (F):

If the request from the client is authorized, the RS fulfills the request and returns a response with the appropriate response code. The RS uses the dynamically established keys to protect the response, according to the communication security protocol used.

The OAuth 2.0 framework defines a number of "protocol flows" via grant types, which have been extended further with extensions to OAuth 2.0 (such as [RFC7521] and [RFC8628]). What grant type works best depends on the usage scenario and [RFC7744] describes many different IoT use cases but there are two grant types that cover a majority of these scenarios, namely the Authorization Code Grant (described in Section 4.1 of [RFC7521]) and the Client Credentials Grant (described in Section 4.4 of [RFC7521]). The Authorization Code Grant is a good fit for use with apps running on smart phones and tablets that request access to IoT devices, a common scenario in the smart home environment, where users need to go through an authentication and authorization phase (at least during the initial setup phase). The native apps guidelines described in [RFC8252] are applicable to this use case. The Client Credential Grant is a good fit for use with IoT devices where the OAuth client itself is constrained. In such a case, the resource owner has pre-arranged access rights for the client with the authorization server, which is often accomplished using a commissioning tool.

The consent of the resource owner, for giving a client access to a protected resource, can be provided dynamically as in the traditional OAuth flows, or it could be pre-configured by the resource owner as authorization policies at the AS, which the AS evaluates when a token request arrives. The resource owner and the requesting party (i.e., client owner) are not shown in Figure 1.

This framework supports a wide variety of communication security mechanisms between the ACE entities, such as client, AS, and RS. It is assumed that the client has been registered (also called enrolled or onboarded) to an AS using a mechanism defined outside the scope of

this document. In practice, various techniques for onboarding have been used, such as factory-based provisioning or the use of commissioning tools. Regardless of the onboarding technique, this provisioning procedure implies that the client and the AS exchange credentials and configuration parameters. These credentials are used to mutually authenticate each other and to protect messages exchanged between the client and the AS.

It is also assumed that the RS has been registered with the AS, potentially in a similar way as the client has been registered with the AS. Established keying material between the AS and the RS allows the AS to apply cryptographic protection to the access token to ensure that its content cannot be modified, and if needed, that the content is confidentiality protected. Confidentiality protection of the access token content would be provided on top of confidentiality protection via a communication security protocol.

The keying material necessary for establishing communication security between C and RS is dynamically established as part of the protocol described in this document.

At the start of the protocol, there is an optional discovery step where the client discovers the resource server and the resources this server hosts. In this step, the client might also determine what permissions are needed to access the protected resource. A generic procedure is described in Section 5.1; profiles MAY define other procedures for discovery.

In Bluetooth Low Energy, for example, advertisements are broadcast by a peripheral, including information about the primary services. In CoAP, as a second example, a client can make a request to `"/.well-known/core"` to obtain information about available resources, which are returned in a standardized format as described in [RFC6690].

5. Framework

The following sections detail the profiling and extensions of OAuth 2.0 for constrained environments, which constitutes the ACE framework.

Credential Provisioning

In constrained environments it cannot be assumed that the client and the RS are part of a common key infrastructure. Therefore, the AS provisions credentials and associated information to allow mutual authentication between the client and the RS. The resulting security association between the client and the RS may then also be used to bind these credentials to the access tokens the client uses.

Proof-of-Possession

The ACE framework, by default, implements proof-of-possession for access tokens, i.e., that the token holder can prove being a holder of the key bound to the token. The binding is provided by the "cnf" claim [RFC8747] indicating what key is used for proof-of-possession. If a client needs to submit a new access token, e.g., to obtain additional access rights, they can request that the AS binds this token to the same key as the previous one.

ACE Profiles

The client or RS may be limited in the encodings or protocols it supports. To support a variety of different deployment settings, specific interactions between client and RS are defined in an ACE profile. In ACE framework the AS is expected to manage the matching of compatible profile choices between a client and an RS. The AS informs the client of the selected profile using the "ace_profile" parameter in the token response.

OAuth 2.0 requires the use of TLS both to protect the communication between AS and client when requesting an access token; between client and RS when accessing a resource and between AS and RS if introspection is used. In constrained settings TLS is not always feasible, or desirable. Nevertheless it is REQUIRED that the communications named above are encrypted, integrity protected and protected against message replay. It is also REQUIRED that the communicating endpoints perform mutual authentication. Furthermore it MUST be assured that responses are bound to the requests in the sense that the receiver of a response can be certain that the response actually belongs to a certain request. Note that setting up such a secure communication may require some unprotected messages to be exchanged first (e.g. sending the token from the client to the RS).

Profiles MUST specify a communication security protocol between client and RS that provides the features required above. Profiles MUST specify a communication security protocol RECOMMENDED to be used between client and AS that provides the features required above. Profiles MUST specify for introspection a communication security protocol RECOMMENDED to be used between RS and AS that provides the features required above. These recommendations enable interoperability between different implementations without the need to define a new profile if the communication between C and AS, or between RS and AS, is protected with a different security protocol complying with the security requirements above.

In OAuth 2.0 the communication with the Token and the Introspection endpoints at the AS is assumed to be via HTTP and may use Uri-query parameters. When profiles of this framework use CoAP instead, it is REQUIRED to use one of the following alternatives instead of Uri-query parameters: The sender (client or RS) encodes the parameters of its request as a CBOR map and submits that map as the payload of the POST request. The CBOR encoding for a number of OAuth 2.0 parameters is specified in this document, if a profile needs to use other OAuth 2.0 parameters with CoAP it MUST specify their CBOR encoding.

Profiles that use CBOR encoding of protocol message parameters at the outermost encoding layer MUST use the content format 'application/ace+cbor'. If CoAP is used for communication, the Content-Format MUST be abbreviated with the ID: 19 (see Section 8.16).

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. If CoAP is used, it is REQUIRED to use CBOR [RFC8949] instead of JSON. Depending on the profile, the CBOR payload MAY be enclosed in a non-CBOR cryptographic wrapper.

5.1. Discovering Authorization Servers

C must discover the AS in charge of RS to determine where to request the access token. To do so, C must 1. find out the AS URI to which the token request message must be sent and 2. MUST validate that the AS with this URI is authorized to provide access tokens for this RS.

In order to determine the AS URI, C MAY send an initial Unauthorized Resource Request message to RS. RS then denies the request and sends the address of its AS back to C (see Section 5.2). How C validates the AS authorization is not in scope for this document. C may, e.g., ask its owner if this AS is authorized for this RS. C may also use a mechanism that addresses both problems at once (e.g. by querying a dedicated secure service provided by the client owner) .

5.2. Unauthorized Resource Request Message

An Unauthorized Resource Request message is a request for any resource hosted by RS for which the client does not have authorization granted. RSes MUST treat any request for a protected resource as an Unauthorized Resource Request message when any of the following hold:

- * The request has been received on an unsecured channel.
- * The RS has no valid access token for the sender of the request regarding the requested action on that resource.

- * The RS has a valid access token for the sender of the request, but that token does not authorize the requested action on the requested resource.

Note: These conditions ensure that the RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The `authz-info` endpoint, as part of the process for authorizing to protected resources, is not itself a protected resource and MUST NOT be protected as specified above (cf. Section 5.10.1).

Unauthorized Resource Request messages MUST be denied with an `"unauthorized_client"` error response. In this response, the Resource Server SHOULD provide proper `"AS Request Creation Hints"` to enable the client to request an access token from RS's AS as described in Section 5.3.

The handling of all client requests (including unauthorized ones) by the RS is described in Section 5.10.2.

5.3. AS Request Creation Hints

The `"AS Request Creation Hints"` message is sent by an RS as a response to an Unauthorized Resource Request message (see Section 5.2) to help the sender of the Unauthorized Resource Request message acquire a valid access token. The `"AS Request Creation Hints"` message is a CBOR or JSON map, with an OPTIONAL element `"AS"` specifying an absolute URI (see Section 4.3 of [RFC3986]) that identifies the appropriate AS for the RS.

The message can also contain the following OPTIONAL parameters:

- * A `"audience"` element contains an identifier the client should request at the AS, as suggested by the RS. With this parameter, when included in the access token request to the AS, the AS is able to restrict the use of access token to specific RSs. See Section 6.9 for a discussion of this parameter.
- * A `"kid"` element containing the key identifier of a key used in an existing security association between the client and the RS. The RS expects the client to request an access token bound to this key, in order to avoid having to re-establish the security association.
- * A `"cnonce"` element containing a client-nonce. See Section 5.3.1.
- * A `"scope"` element containing the suggested scope that the client should request towards the AS.

Figure 2 summarizes the parameters that may be part of the "AS Request Creation Hints".

Name	CBOR Key	Value Type
AS	1	text string
kid	2	byte string
audience	5	text string
scope	9	text or byte string
cnonce	39	byte string

Figure 2: AS Request Creation Hints

Note that the schema part of the AS parameter may need to be adapted to the security protocol that is used between the client and the AS. Thus the example AS value "coap://as.example.com/token" might need to be transformed to "coaps://as.example.com/token". It is assumed that the client can determine the correct schema part on its own depending on the way it communicates with the AS.

Figure 3 shows an example for an "AS Request Creation Hints" message payload using CBOR [RFC8949] diagnostic notation, using the parameter names instead of the CBOR keys for better human readability.

```
4.01 Unauthorized
Content-Format: application/ace+cbor
Payload :
{
  "AS" : "coaps://as.example.com/token",
  "audience" : "coaps://rs.example.com"
  "scope" : "rTempC",
  "cnonce" : h'e0a156bb3f'
}
```

Figure 3: AS Request Creation Hints payload example

In the example above, the response parameter "AS" points the receiver of this message to the URI "coaps://as.example.com/token" to request access tokens. The RS sending this response uses an internal clock that is not synchronized with the clock of the AS. Therefore, it can not reliably verify the expiration time of access tokens it receives. To ensure a certain level of access token freshness nevertheless, the RS has included a cnonce parameter (see Section 5.3.1) in the response. (The hex-sequence of the cnonce parameter is encoded in CBOR-based notation in this example.)

Figure 4 illustrates the mandatory to use binary encoding of the message payload shown in Figure 3.

```

a4                                # map(4)
 01                                # unsigned(1) (=AS)
 78 1c                            # text(28)
    636f6170733a2f2f61732e657861
    6d706c652e636f6d2f746f6b656e    # "coaps://as.example.com/token"
 05                                # unsigned(5) (=audience)
 76                                # text(22)
    636f6170733a2f2f72732e657861
    6d706c652e636f6d                # "coaps://rs.example.com"
 09                                # unsigned(9) (=scope)
 66                                # text(6)
    72546556d7043                  # "rTempC"
 18 27                            # unsigned(39) (=cnonce)
 45                                # bytes(5)
    e0a156bb3f                     #

```

Figure 4: AS Request Creation Hints example encoded in CBOR

5.3.1. The Client-Nonce Parameter

If the RS does not synchronize its clock with the AS, it could be tricked into accepting old access tokens, that are either expired or have been compromised. In order to ensure some level of token freshness in that case, the RS can use the "cnonce" (client-nonce) parameter. The processing requirements for this parameter are as follows:

- * An RS sending a "cnonce" parameter in an "AS Request Creation Hints" message MUST store information to validate that a given cnonce is fresh. How this is implemented internally is out of scope for this specification. Expiration of client-nonces should be based roughly on the time it would take a client to obtain an access token after receiving the "AS Request Creation Hints" message, with some allowance for unexpected delays.
- * A client receiving a "cnonce" parameter in an "AS Request Creation Hints" message MUST include this in the parameters when requesting an access token at the AS, using the "cnonce" parameter from Section 5.8.4.4.
- * If an AS grants an access token request containing a "cnonce" parameter, it MUST include this value in the access token, using the "cnonce" claim specified in Section 5.10.

- * An RS that is using the client-nonce mechanism and that receives an access token MUST verify that this token contains a cnonce claim, with a client-nonce value that is fresh according to the information stored at the first step above. If the cnonce claim is not present or if the cnonce claim value is not fresh, the RS MUST discard the access token. If this was an interaction with the authz-info endpoint the RS MUST also respond with an error message using a response code equivalent to the CoAP code 4.01 (Unauthorized).

5.4. Authorization Grants

To request an access token, the client obtains authorization from the resource owner or uses its client credentials as a grant. The authorization is expressed in the form of an authorization grant.

The OAuth framework [RFC6749] defines four grant types. The grant types can be split up into two groups, those granted on behalf of the resource owner (password, authorization code, implicit) and those for the client (client credentials). Further grant types have been added later, such as [RFC7521] defining an assertion-based authorization grant.

The grant type is selected depending on the use case. In cases where the client acts on behalf of the resource owner, the authorization code grant is recommended. If the client acts on behalf of the resource owner, but does not have any display or has very limited interaction possibilities, it is recommended to use the device code grant defined in [RFC8628]. In cases where the client acts autonomously the client credentials grant is recommended.

For details on the different grant types, see section 1.3 of [RFC6749]. The OAuth 2.0 framework provides an extension mechanism for defining additional grant types, so profiles of this framework MAY define additional grant types, if needed.

5.5. Client Credentials

Authentication of the client is mandatory independent of the grant type when requesting an access token from the token endpoint. In the case of the client credentials grant type, the authentication and grant coincide.

Client registration and provisioning of client credentials to the client is out of scope for this specification.

The OAuth framework defines one client credential type in section 2.3.1 of [RFC6749]: client id and client secret.

[I-D.erdman-ace-rpcc] adds raw-public-key and pre-shared-key to the client credentials types. Profiles of this framework MAY extend with an additional client credentials type using client certificates.

5.6. AS Authentication

The client credential grant does not, by default, authenticate the AS that the client connects to. In classic OAuth, the AS is authenticated with a TLS server certificate.

Profiles of this framework MUST specify how clients authenticate the AS and how communication security is implemented. By default, server side TLS certificates, as defined by OAuth 2.0, are required.

5.7. The Authorization Endpoint

The OAuth 2.0 authorization endpoint is used to interact with the resource owner and obtain an authorization grant, in certain grant flows. The primary use case for the ACE-OAuth framework is for machine-to-machine interactions that do not involve the resource owner in the authorization flow; therefore, this endpoint is out of scope here. Future profiles may define constrained adaptation mechanisms for this endpoint as well. Non-constrained clients interacting with constrained resource servers can use the specification in section 3.1 of [RFC6749] and the attack countermeasures suggested in section 4.2 of [RFC6819].

5.8. The Token Endpoint

In standard OAuth 2.0, the AS provides the token endpoint for submitting access token requests. This framework extends the functionality of the token endpoint, giving the AS the possibility to help the client and RS to establish shared keys or to exchange their public keys. Furthermore, this framework defines encodings using CBOR, as a substitute for JSON.

The endpoint may also be exposed over HTTPS as in classical OAuth or even other transports. A profile MUST define the details of the mapping between the fields described below, and these transports. If HTTPS is used, the semantics of Sections 4.1.3 and 4.1.4 of the OAuth 2.0 specification MUST be followed (with additions as described below). If the CoAP is some other transport with CBOR payload format is supported, the semantics described in this section MUST be followed.

For the AS to be able to issue a token, the client MUST be authenticated and present a valid grant for the scopes requested. Profiles of this framework MUST specify how the AS authenticates the client and how the communication between client and AS is protected, fulfilling the requirements specified in Section 5.

The default name of this endpoint in an url-path SHOULD be `'/token'`. However, implementations are not required to use this name and can define their own instead.

The figures of this section use CBOR diagnostic notation without the integer abbreviations for the parameters or their values for illustrative purposes. Note that implementations MUST use the integer abbreviations and the binary CBOR encoding, if the CBOR encoding is used.

5.8.1. Client-to-AS Request

The client sends a POST request to the token endpoint at the AS. The profile MUST specify how the communication is protected. The content of the request consists of the parameters specified in the relevant subsection of section 4 of the OAuth 2.0 specification [RFC6749], depending on the grant type, with the following exceptions and additions:

- * The parameter `"grant_type"` is OPTIONAL in the context of this framework (as opposed to REQUIRED in RFC6749). If that parameter is missing, the default value `"client_credentials"` is implied.
- * The `"audience"` parameter from [RFC8693] is OPTIONAL to request an access token bound to a specific audience.
- * The `"cnonce"` parameter defined in Section 5.8.4.4 is REQUIRED if the RS provided a client-nonce in the `"AS Request Creation Hints"` message Section 5.3
- * The `"scope"` parameter MAY be encoded as a byte string instead of the string encoding specified in section 3.3 of [RFC6749], in order allow compact encoding of complex scopes. The syntax of such a binary encoding is explicitly not specified here and left to profiles or applications. Note specifically that a binary encoded scope does not necessarily use the space character `'0x20'` to delimit scope-tokens.
- * The client can send an empty (null value) `"ace_profile"` parameter to indicate that it wants the AS to include the `"ace_profile"` parameter in the response. See Section 5.8.4.3.

- * A client MUST be able to use the parameters from [I-D.ietf-ace-oauth-params] in an access token request to the token endpoint and the AS MUST be able to process these additional parameters.

The default behavior, is that the AS generates a symmetric proof-of-possession key for the client. In order to use an asymmetric key pair or to re-use a key previously established with the RS, the client is supposed to use the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

If CoAP is used then these parameters MUST be provided in a CBOR map, see Figure 12.

When HTTP is used as a transport then the client makes a request to the token endpoint, the parameters MUST be encoded as defined in Appendix B of [RFC6749].

The following examples illustrate different types of requests for proof-of-possession tokens.

Figure 5 shows a request for a token with a symmetric proof-of-possession key. The content is displayed in CBOR diagnostic notation, without abbreviations for better readability.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "audience" : "tempSensor4711"
}
```

Figure 5: Example request for an access token bound to a symmetric key.

Figure 6 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example OSCORE [RFC8613] is used to provide object-security, therefore the Content-Format is "application/oscore" wrapping the "application/ace+cbor" type content. The OSCORE option has a decoded interpretation appended in parentheses for the reader's convenience. Also note that in this example the audience is implicitly known by both client and AS. Furthermore note that this example uses the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].


```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
OSCORE: 0x09, 0x05, 0x44, 0x6C
      (h=0, k=1, n=001, partialIV= 0x05, kid=[0x44, 0x6C])
Content-Format: "application/oscore"
Payload:
  0x44025d1 ... (full payload omitted for brevity) ... 68b3825e
```

Decrypted payload:

```
{
  "client_id" : "myclient",
  "req_cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h'11',
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
    }
  }
}
```

Figure 6: Example token request bound to an asymmetric key.

Figure 7 shows a request for a token where a previously communicated proof-of-possession key is only referenced using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "audience" : "valve424",
  "scope" : "read",
  "req_cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

Figure 7: Example request for an access token bound to a key reference.

Refresh tokens are typically not stored as securely as proof-of-possession keys in requesting clients. Proof-of-possession based refresh token requests MUST NOT request different proof-of-possession keys or different audiences in token requests. Refresh token requests can only use to request access tokens bound to the same proof-of-possession key and the same audience as access tokens issued in the initial token request.

5.8.2. AS-to-Client Response

If the access token request has been successfully verified by the AS and the client is authorized to obtain an access token corresponding to its access token request, the AS sends a response with the response code equivalent to the CoAP response code 2.01 (Created). If client request was invalid, or not authorized, the AS returns an error response as described in Section 5.8.3.

Note that the AS decides which token type and profile to use when issuing a successful response. It is assumed that the AS has prior knowledge of the capabilities of the client and the RS (see Appendix D). This prior knowledge may, for example, be set by the use of a dynamic client registration protocol exchange [RFC7591]. If the client has requested a specific proof-of-possession key using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params], this may also influence which profile the AS selects, as it needs to support the use of the key type requested the client.

The content of the successful reply is the Access Information. When using CoAP, the payload MUST be encoded as a CBOR map, when using HTTP the encoding is a JSON map as specified in section 5.1 of [RFC6749]. In both cases the parameters specified in Section 5.1 of [RFC6749] are used, with the following additions and changes:

ace_profile:

OPTIONAL unless the request included an empty ace_profile parameter in which case it is MANDATORY. This indicates the profile that the client MUST use towards the RS. See Section 5.8.4.3 for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

token_type:

This parameter is OPTIONAL, as opposed to 'required' in [RFC6749]. By default implementations of this framework SHOULD assume that the token_type is "PoP". If a specific use case requires another token_type (e.g., "Bearer") to be used then this parameter is REQUIRED.

Furthermore [I-D.ietf-ace-oauth-params] defines additional parameters that the AS MUST be able to use when responding to a request to the token endpoint.

Figure 8 summarizes the parameters that can currently be part of the Access Information. Future extensions may define additional parameters.

Parameter name	Specified in
access_token	RFC 6749
token_type	RFC 6749
expires_in	RFC 6749
refresh_token	RFC 6749
scope	RFC 6749
state	RFC 6749
error	RFC 6749
error_description	RFC 6749
error_uri	RFC 6749
ace_profile	[this document]
cnf	[I-D.ietf-ace-oauth-params]
rs_cnf	[I-D.ietf-ace-oauth-params]

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key, which is defined in [I-D.ietf-ace-oauth-params]. Note that the key identifier 'kid' is only used to simplify indexing and retrieving the key, and no assumptions should be made that it is unique in the domains of either the client or the RS.

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
    (remainder of CWT omitted for brevity;
    CWT contains COSE_Key in the "cnf" claim)',
  "ace_profile" : "coap_dtls",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 9: Example AS response with an access token bound to a symmetric key.

5.8.3. Error Response

The error responses for interactions with the AS are generally equivalent to the ones defined in Section 5.2 of [RFC6749], with the following exceptions:

- * When using CoAP the payload MUST be encoded as a CBOR map, with the Content-Format "application/ace+cbor". When using HTTP the payload is encoded in JSON as specified in section 5.2 of [RFC6749].
- * A response code equivalent to the CoAP code 4.00 (Bad Request) MUST be used for all error responses, except for `invalid_client` where a response code equivalent to the CoAP code 4.01 (Unauthorized) MAY be used under the same conditions as specified in Section 5.2 of [RFC6749].
- * The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12, when a CBOR encoding is used.
- * The error code (i.e., value of the "error" parameter) MUST be abbreviated as specified in Figure 10, when a CBOR encoding is used.

Name	CBOR Values	Original Specification
invalid_request	1	section 5.2 of [RFC6749]
invalid_client	2	section 5.2 of [RFC6749]
invalid_grant	3	section 5.2 of [RFC6749]
unauthorized_client	4	section 5.2 of [RFC6749]
unsupported_grant_type	5	section 5.2 of [RFC6749]
invalid_scope	6	section 5.2 of [RFC6749]
unsupported_pop_key	7	[this document]
incompatible_ace_profiles	8	[this document]

Figure 10: CBOR abbreviations for common error codes

In addition to the error responses defined in OAuth 2.0, the following behavior MUST be implemented by the AS:

- * If the client submits an asymmetric key in the token request that the RS cannot process, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "unsupported_pop_key" specified in Figure 10.
- * If the client and the RS it has requested an access token for do not share a common profile, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "incompatible_ace_profiles" specified in Figure 10.

5.8.4. Request and Response Parameters

This section provides more detail about the new parameters that can be used in access token requests and responses, as well as abbreviations for more compact encoding of existing parameters and common parameter values.

5.8.4.1. Grant Type

The abbreviations specified in the registry defined in Section 8.5 MUST be used in CBOR encodings instead of the string values defined in [RFC6749], if CBOR payloads are used.

Name	CBOR Value	Original Specification
password	0	s. 4.3.2 of [RFC6749]
authorization_code	1	s. 4.1.3 of [RFC6749]
client_credentials	2	s. 4.4.2 of [RFC6749]
refresh_token	3	s. 6 of [RFC6749]

Figure 11: CBOR abbreviations for common grant types

5.8.4.2. Token Type

The "token_type" parameter, defined in section 5.1 of [RFC6749], allows the AS to indicate to the client which type of access token it is receiving (e.g., a bearer token).

This document registers the new value "PoP" for the OAuth Access Token Types registry, specifying a proof-of-possession token. How the proof-of-possession by the client to the RS is performed MUST be specified by the profiles.

The values in the "token_type" parameter MUST use the CBOR abbreviations defined in the registry specified by Section 8.7, if a CBOR encoding is used.

In this framework the "pop" value for the "token_type" parameter is the default. The AS may, however, provide a different value from those registered in [IANA.OAuthAccessTokenTypes].

5.8.4.3. Profile

Profiles of this framework MUST define the communication protocol and the communication security protocol between the client and the RS. The security protocol MUST provide encryption, integrity and replay protection. It MUST also provide a binding between requests and responses. Furthermore profiles MUST define a list of allowed proof-of-possession methods, if they support proof-of-possession tokens.

A profile MUST specify an identifier that MUST be used to uniquely identify itself in the "ace_profile" parameter. The textual representation of the profile identifier is intended for human readability and for JSON-based interactions, it MUST NOT be used for CBOR-based interactions. Profiles MUST register their identifier in the registry defined in Section 8.8.

Profiles MAY define additional parameters for both the token request and the Access Information in the access token response in order to support negotiation or signaling of profile specific parameters.

Clients that want the AS to provide them with the "ace_profile" parameter in the access token response can indicate that by sending a ace_profile parameter with a null value for CBOR-based interactions, or an empty string if CBOR is not used, in the access token request.

5.8.4.4. Client-Nonce

This parameter MUST be sent from the client to the AS, if it previously received a "cnonce" parameter in the "AS Request Creation Hints" Section 5.3. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (base64url without padding encoded binary [RFC4648]) if CBOR is not used. It MUST copy the value from the cnonce parameter in the "AS Request Creation Hints".

5.8.5. Mapping Parameters to CBOR

If CBOR encoding is used, all OAuth parameters in access token requests and responses MUST be mapped to CBOR types as specified in the registry defined by Section 8.10, using the given integer abbreviation for the map keys.

Note that we have aligned the abbreviations corresponding to claims with the abbreviations defined in [RFC8392].

Note also that abbreviations from -24 to 23 have a 1 byte encoding size in CBOR. We have thus chosen to assign abbreviations in that range to parameters we expect to be used most frequently in constrained scenarios.

Name	CBOR Key	Value Type	Original Specification
access_token	1	byte string	[RFC6749]
expires_in	2	unsigned integer	[RFC6749]
audience	5	text string	[RFC8693]
scope	9	text or byte string	[RFC6749]
client_id	24	text string	[RFC6749]
client_secret	25	byte string	[RFC6749]
response_type	26	text string	[RFC6749]
redirect_uri	27	text string	[RFC6749]
state	28	text string	[RFC6749]
code	29	byte string	[RFC6749]
error	30	integer	[RFC6749]
error_description	31	text string	[RFC6749]
error_uri	32	text string	[RFC6749]
grant_type	33	unsigned integer	[RFC6749]
token_type	34	integer	[RFC6749]
username	35	text string	[RFC6749]
password	36	text string	[RFC6749]
refresh_token	37	byte string	[RFC6749]
ace_profile	38	integer	[this document]
cnonce	39	byte string	[this document]

Figure 12: CBOR mappings used in token requests and responses

5.9. The Introspection Endpoint

Token introspection [RFC7662] MAY be implemented by the AS, and the RS. When implemented, it MAY be used by the RS and to query the AS for metadata about a given token, e.g., validity or scope. Analogous to the protocol defined in [RFC7662] for HTTP and JSON, this section defines adaptations to more constrained environments using CBOR and leaving the choice of the application protocol to the profile.

Communication between the requesting entity and the introspection endpoint at the AS MUST be integrity protected and encrypted. The communication security protocol MUST also provide a binding between requests and responses. Furthermore, the two interacting parties MUST perform mutual authentication. Finally, the AS SHOULD verify that the requesting entity has the right to access introspection information about the provided token. Profiles of this framework that support introspection MUST specify how authentication and communication security between the requesting entity and the AS is implemented.

The default name of this endpoint in an url-path SHOULD be `"/introspect"`. However, implementations are not required to use this name and can define their own instead.

The figures of this section use the CBOR diagnostic notation without the integer abbreviations for the parameters and their values for better readability.

5.9.1. Introspection Request

The requesting entity sends a POST request to the introspection endpoint at the AS. The profile MUST specify how the communication is protected. If CoAP is used, the payload MUST be encoded as a CBOR map with a "token" entry containing the access token. Further optional parameters representing additional context that is known by the requesting entity to aid the AS in its response MAY be included.

For CoAP-based interaction, all messages MUST use the content type `"application/ace+cbor"`. For HTTP the encoding defined in section 2.1 of [RFC7662] is used.

The same parameters are required and optional as in Section 2.1 of [RFC7662].

For example, Figure 13 shows an RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that object security based on OSCORE [RFC8613] is assumed in this example, therefore the Content-Format is `"application/oscore"`. Figure 14 shows the decoded payload.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "introspect"
OSCORE: 0x09, 0x05, 0x25
Content-Format: "application/oscore"
Payload:
... COSE content ...
```

Figure 13: Example introspection request.

```
{
  "token" : b64'7gj0dXJQ43U',
  "token_type_hint" : "PoP"
}
```

Figure 14: Decoded payload.

5.9.2. Introspection Response

If the introspection request is authorized and successfully processed, the AS sends a response with the response code equivalent to the CoAP code 2.01 (Created). If the introspection request was invalid, not authorized or couldn't be processed the AS returns an error response as described in Section 5.9.3.

In a successful response, the AS encodes the response parameters in a map. If CoAP is used, this MUST be encoded as a CBOR map, if HTTP is used the JSON encoding specified in section 2.2 of [RFC7662] is used. The map containing the response payload includes the same required and optional parameters as in Section 2.2 of [RFC7662] with the following additions:

`ace_profile` OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.8.4.3 for more details on the formatting of this parameter. If this parameter is absent, the AS assumes that the RS implicitly knows which profile to use towards the client.

`cnonce` OPTIONAL. A client-nonce provided to the AS by the client. The RS MUST verify that this corresponds to the client-nonce previously provided to the client in the "AS Request Creation Hints". See Section 5.3 and Section 5.8.4.4. Its value is a byte string when encoded in CBOR and the base64url encoding of this byte string without padding when encoded in JSON [RFC4648].

`cti` OPTIONAL. The "cti" claim associated to this access token. This parameter has the same meaning and processing rules as the "jti" parameter defined in section 3.1.2 of [RFC7662] except that its value is a byte string when encoded in CBOR and the base64url encoding of this byte string without padding when encoded in JSON [RFC4648].

`exp` OPTIONAL. The "expires-in" claim associated to this access token. See Section 5.10.3.

Furthermore [I-D.ietf-ace-oauth-params] defines more parameters that the AS MUST be able to use when responding to a request to the introspection endpoint.

For example, Figure 15 shows an AS response to the introspection request in Figure 13. Note that this example contains the "cnf" parameter defined in [I-D.ietf-ace-oauth-params].

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "active" : true,
  "scope" : "read",
  "ace_profile" : "coap_dtls",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 15: Example introspection response.

5.9.3. Error Response

The error responses for CoAP-based interactions with the AS are equivalent to the ones for HTTP-based interactions as defined in Section 2.3 of [RFC7662], with the following differences:

- * If content is sent and CoAP is used the payload MUST be encoded as a CBOR map and the Content-Format "application/ace+cbor" MUST be used. For HTTP the encoding defined in section 2.3 of [RFC6749] is used.
- * If the credentials used by the requesting entity (usually the RS) are invalid the AS MUST respond with the response code equivalent to the CoAP code 4.01 (Unauthorized) and use the required and optional parameters from Section 2.3 in [RFC7662].
- * If the requesting entity does not have the right to perform this introspection request, the AS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden). In this case no payload is returned.
- * The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12.
- * The error codes MUST be abbreviated using the codes specified in the registry defined by Section 8.4.

Note that a properly formed and authorized query for an inactive or otherwise invalid token does not warrant an error response by this specification. In these cases, the authorization server MUST instead respond with an introspection response with the "active" field set to "false".

5.9.4. Mapping Introspection Parameters to CBOR

If CBOR is used, the introspection request and response parameters MUST be mapped to CBOR types as specified in the registry defined by Section 8.12, using the given integer abbreviation for the map key.

Note that we have aligned abbreviations that correspond to a claim with the abbreviations defined in [RFC8392] and the abbreviations of parameters with the same name from Section 5.8.5.

Parameter name	CBOR Key	Value Type	Original Specification
iss	1	text string	[RFC7662]
sub	2	text string	[RFC7662]
aud	3	text string	[RFC7662]
exp	4	integer or floating-point number	[RFC7662]
nbf	5	integer or floating-point number	[RFC7662]
iat	6	integer or floating-point number	[RFC7662]
cti	7	byte string	[this document]
scope	9	text or byte string	[RFC7662]
active	10	True or False	[RFC7662]
token	11	byte string	[RFC7662]
client_id	24	text string	[RFC7662]
error	30	integer	[RFC7662]
error_description	31	text string	[RFC7662]
error_uri	32	text string	[RFC7662]
token_type_hint	33	text string	[RFC7662]
token_type	34	integer	[RFC7662]
username	35	text string	[RFC7662]
ace_profile	38	integer	[this document]
cnonce	39	byte string	[this document]
exi	40	unsigned integer	[this document]

Figure 16: CBOR mappings for Token Introspection Parameters.

5.10. The Access Token

In this framework the use of CBOR Web Token (CWT) as specified in [RFC8392] is RECOMMENDED.

In order to facilitate offline processing of access tokens, this document uses the "cnf" claim from [RFC8747] and the "scope" claim from [RFC8693] for JWT- and CWT-encoded tokens. In addition to string encoding specified for the "scope" claim, a binary encoding MAY be used. The syntax of such an encoding is explicitly not specified here and left to profiles or applications, specifically note that a binary encoded scope does not necessarily use the space character '0x20' to delimit scope-tokens.

If the AS needs to convey a hint to the RS about which profile it should use to communicate with the client, the AS MAY include an "ace_profile" claim in the access token, with the same syntax and semantics as defined in Section 5.8.4.3.

If the client submitted a client-nonce parameter in the access token request Section 5.8.4.4, the AS MUST include the value of this parameter in the "cnonce" claim specified here. The "cnonce" claim uses binary encoding.

5.10.1. The Authorization Information Endpoint

The access token, containing authorization information and information about the proof-of-possession method used by the client, needs to be transported to the RS so that the RS can authenticate and authorize the client request.

This section defines a method for transporting the access token to the RS using a RESTful protocol such as CoAP. Profiles of this framework MAY define other methods for token transport.

The method consists of an authz-info endpoint, implemented by the RS. A client using this method MUST make a POST request to the authz-info endpoint at the RS with the access token in the payload. The CoAP Content-Format or HTTP Media Type MUST reflect the format of the token, e.g. application/cwt for CBOR Web Tokens, if no Content-Format or Media Type is defined for the token format, application/octet-stream MUST be used.

The RS receiving the token MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with a response code equivalent to CoAP's 2.01 (Created). Section 5.10.1.1 outlines how an RS MUST proceed to verify the validity of an access token.

The RS MUST be prepared to store at least one access token for future use. This is a difference to how access tokens are handled in OAuth 2.0, where the access token is typically sent along with each request, and therefore not stored at the RS.

When using this framework it is RECOMMENDED that an RS stores only one token per proof-of-possession key. This means that an additional token linked to the same key will supersede any existing token at the RS, by replacing the corresponding authorization information. The reason is that this greatly simplifies (constrained) implementations, with respect to required storage and resolving a request to the applicable token. The use of multiple access tokens for a single client increases the strain on the resource server as it must consider every access token and calculate the actual permissions of the client. Also, tokens may contradict each other which may lead the server to enforce wrong permissions. If one of the access tokens expires earlier than others, the resulting permissions may offer insufficient protection.

If the payload sent to the authz-info endpoint does not parse to a token, the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request).

The RS MAY make an introspection request to validate the token before responding to the POST request to the authz-info endpoint, e.g. if the token is an opaque reference. Some transport protocols may provide a way to indicate that the RS is busy and the client should retry after an interval; this type of status update would be appropriate while the RS is waiting for an introspection response.

Profiles MUST specify whether the authz-info endpoint is protected, including whether error responses from this endpoint are protected. Note that since the token contains information that allow the client and the RS to establish a security context in the first place, mutual authentication may not be possible at this point.

The default name of this endpoint in an url-path is '/authz-info', however implementations are not required to use this name and can define their own instead.

5.10.1.1. Verifying an Access Token

When an RS receives an access token, it MUST verify it before storing it. The details of token verification depends on various aspects, including the token encoding, the type of token, the security protection applied to the token, and the claims. The token encoding matters since the security protection differs between the token encodings. For example, a CWT token uses COSE while a JWT token uses JOSE. The type of token also has an influence on the verification procedure since tokens may be self-contained whereby token verification may happen locally at the RS while a token-by-reference requires further interaction with the authorization server, for example using token introspection, to obtain the claims associated with the token reference. Self-contained tokens MUST, at least be integrity protected but they MAY also be encrypted.

For self-contained tokens the RS MUST process the security protection of the token first, as specified by the respective token format. For CWT the description can be found in [RFC8392] and for JWT the relevant specification is [RFC7519]. This MUST include a verification that security protection (and thus the token) was generated by an AS that has the right to issue access tokens for this RS.

In case the token is communicated by reference the RS needs to obtain the claims first. When the RS uses token introspection the relevant specification is [RFC7662] with CoAP transport specified in Section 5.9.

Errors may happen during this initial processing stage:

- * If the verification of the security wrapper fails, or the token was issued by an AS that does not have the right to issue tokens for the receiving RS, the RS MUST discard the token and, if this was an interaction with authz-info, return an error message with a response code equivalent to the CoAP code 4.01 (Unauthorized).
- * If the claims cannot be obtained the RS MUST discard the token and, in case of an interaction via the authz-info endpoint, return an error message with a response code equivalent to the CoAP code 4.00 (Bad Request).

Next, the RS MUST verify claims, if present, contained in the access token. Errors are returned when claim checks fail, in the order of priority of this list:

iss The issuer claim (if present) must identify the AS that has

produced the security protection for the access token. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.01 (Unauthorized).

exp The expiration date must be in the future. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info the RS MUST also respond with a response code equivalent to the CoAP code 4.01 (Unauthorized). Note that the RS has to terminate access rights to the protected resources at the time when the tokens expire.

aud The audience claim must refer to an audience that the RS identifies with. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.03 (Forbidden).

scope The RS must recognize value of the scope claim. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.00 (Bad Request). The RS MAY provide additional information in the error response, to clarify what went wrong.

Additional processing may be needed for other claims in a way specific to a profile or the underlying application.

Note that the Subject (sub) claim cannot always be verified when the token is submitted to the RS since the client may not have authenticated yet. Also note that a counter for the expires_in (exp) claim MUST be initialized when the RS first verifies this token.

Also note that profiles of this framework may define access token transport mechanisms that do not allow for error responses. Therefore the error messages specified here only apply if the token was sent to the authz-info endpoint.

When sending error responses, the RS MAY use the error codes from Section 3.1 of [RFC6750], to provide additional details to the client.

5.10.1.2. Protecting the Authorization Information Endpoint

As this framework can be used in RESTful environments, it is important to make sure that attackers cannot perform unauthorized requests on the authz-info endpoints, other than submitting access tokens.

Specifically it SHOULD NOT be possible to perform GET, DELETE or PUT on the authz-info endpoint.

The RS SHOULD implement rate limiting measures to mitigate attacks aiming to overload the processing capacity of the RS by repeatedly submitting tokens. For CoAP-based communication the RS could use the mechanisms from [RFC8516] to indicate that it is overloaded.

5.10.2. Client Requests to the RS

Before sending a request to an RS, the client MUST verify that the keys used to protect this communication are still valid. See Section 5.10.4 for details on how the client determines the validity of the keys used.

If an RS receives a request from a client, and the target resource requires authorization, the RS MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession binding that token to the request.

The response code MUST be 4.01 (Unauthorized) in case the client has not performed the proof-of-possession, or if RS has no valid access token for the client. If RS has an access token for the client but the token does not authorize access for the resource that was requested, RS MUST reject the request with a 4.03 (Forbidden). If RS has an access token for the client but it does not cover the action that was requested on the resource, RS MUST reject the request with a 4.05 (Method Not Allowed).

Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a dumb client optimistically tries to access a requested resource with any access token received from AS. As malicious clients could pretend to be C to determine C's privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the client is authenticated.

Note: The RS MAY use introspection for timely validation of an access token, at the time when a request is presented.

Note: Matching the claims of the access token (e.g., scope) to a specific request is application specific.

If the request matches a valid token and the client has performed the proof-of-possession for that token, the RS continues to process the request as specified by the underlying application.

5.10.3. Token Expiration

Depending on the capabilities of the RS, there are various ways in which it can verify the expiration of a received access token. Here follows a list of the possibilities including what functionality they require of the RS.

- * The token is a CWT and includes an "exp" claim and possibly the "nbf" claim. The RS verifies these by comparing them to values from its internal clock as defined in [RFC7519]. In this case the RS's internal clock must reflect the current date and time, or at least be synchronized with the AS's clock. How this clock synchronization would be performed is out of scope for this specification.
- * The RS verifies the validity of the token by performing an introspection request as specified in Section 5.9. This requires the RS to have a reliable network connection to the AS and to be able to handle two secure sessions in parallel (C to RS and RS to AS).
- * In order to support token expiration for devices that have no reliable way of synchronizing their internal clocks, this specification defines the following approach: The claim "exp" ("expires in") can be used, to provide the RS with the lifetime of the token in seconds from the time the RS first receives the token. This mechanism only works for self-contained tokens, i.e. CWTs and JWTs. For CWTs this parameter is encoded as unsigned integer, while JWTs encode this as JSON number.
- * Processing this claim requires that the RS does the following:
 - For each token the RS receives, that contains an "exp" claim: Keep track of the time it received that token and revisit that list regularly to expunge expired tokens.
 - Keep track of the identifiers of tokens containing the "exp" claim that have expired (in order to avoid accepting them again). In order to avoid an unbounded memory usage growth, this MUST be implemented in the following way when the "exp" claim is used:
 - o When creating the token, the AS MUST add a 'cti' claim (or 'jti' for JWTs) to the access token. The value of this claim MUST be created as the binary representation of the concatenation of the identifier of the RS with a sequence number counting the tokens containing an 'exp' claim, issued by this AS for the RS.

- o The RS MUST store the highest sequence number of an expired token containing the "exp" claim that it has seen, and treat tokens with lower sequence numbers as expired. Note that this could lead to discarding valid tokens with lower sequence numbers, if the AS were to issue tokens of different validity time for the same RS. The assumption is that typically tokens in such a scenario would all have the same validity time.

If a token that authorizes a long running request such as a CoAP Observe [RFC7641] expires, the RS MUST send an error response with the response code equivalent to the CoAP code 4.01 (Unauthorized) to the client and then terminate processing the long running request.

5.10.4. Key Expiration

The AS provides the client with key material that the RS uses. This can either be a common symmetric PoP-key, or an asymmetric key used by the RS to authenticate towards the client. Since there is currently no expiration metadata associated to those keys, the client has no way of knowing if these keys are still valid. This may lead to situations where the client sends requests containing sensitive information to the RS using a key that is expired and possibly in the hands of an attacker, or accepts responses from the RS that are not properly protected and could possibly have been forged by an attacker.

In order to prevent this, the client must assume that those keys are only valid as long as the related access token is. Since the access token is opaque to the client, one of the following methods MUST be used to inform the client about the validity of an access token:

- * The client knows a default validity time for all tokens it is using (i.e. how long a token is valid after being issued). This information could be provisioned to the client when it is registered at the AS, or published by the AS in a way that the client can query.
- * The AS informs the client about the token validity using the "expires_in" parameter in the Access Information.

A client that is not able to obtain information about the expiration of a token MUST NOT use this token.

6. Security Considerations

Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work. Furthermore [RFC6819] provides additional security considerations for OAuth which apply to IoT deployments as well. If the introspection endpoint is used, the security considerations from [RFC7662] also apply.

The following subsections address issues specific to this document and its use in constrained environments.

6.1. Protecting Tokens

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest (MAC) or an Authenticated Encryption with Associated Data (AEAD) algorithm. Consequently, the token integrity protection **MUST** be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key used for proof-of-possession. If the access token contains the symmetric key, this symmetric key **MUST** be encrypted by the authorization server so that only the resource server can decrypt it. Note that using an AEAD algorithm is preferable over using a MAC unless the token needs to be publicly readable.

If the token is intended for multiple recipients (i.e. an audience that is a group), integrity protection of the token with a symmetric key, shared between the AS and the recipients, is not sufficient, since any of the recipients could modify the token undetected by the other recipients. Therefore a token with a multi-recipient audience **MUST** be protected with an asymmetric signature.

It is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. The same shared secret **MUST NOT** be used as proof-of-possession key with multiple resource servers since the benefit from using the proof-of-possession concept is then significantly reduced.

If clients are capable of doing so, they should frequently request fresh access tokens, as this allows the AS to keep the lifetime of the tokens short. This allows the AS to use shorter proof-of-possession key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens, they SHOULD scope these access tokens to a specific permission.

In certain situations it may be necessary to revoke an access token that is still valid. Client-initiated revocation is specified in [RFC7009] for OAuth 2.0. Other revocation mechanisms are currently not specified, as the underlying assumption in OAuth is that access tokens are issued with a relatively short lifetime. This may not hold true for disconnected constrained devices, needing access tokens with relatively long lifetimes, and would therefore necessitate further standardization work that is out of scope for this document.

6.2. Communication Security

Communication with the authorization server MUST use confidentiality protection. This step is extremely important since the client or the RS may obtain the proof-of-possession key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby completely negating proof-of-possession security. The requirements for communication security of profiles are specified in Section 5.

Additional protection for the access token can be applied by encrypting it, for example encryption of CWTs is specified in Section 5.1 of [RFC8392]. Such additional protection can be necessary if the token is later transferred over an insecure connection (e.g. when it is sent to the authz-info endpoint).

Care must be taken by developers to prevent leakage of the PoP credentials (i.e., the private key or the symmetric key). An adversary in possession of the PoP credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many constrained environments, since adversaries may get physical access to the devices and can therefore use physical extraction techniques to gain access to memory contents. This risk can be mitigated to some extent by making sure that keys are refreshed frequently, by using software isolation techniques and by using hardware security.

6.3. Long-Term Credentials

Both clients and RSs have long-term credentials that are used to secure communications, and authenticate to the AS. These credentials need to be protected against unauthorized access. In constrained devices, deployed in publicly accessible places, such protection can be difficult to achieve without specialized hardware (e.g. secure key storage memory).

If credentials are lost or compromised, the operator of the affected devices needs to have procedures to invalidate any access these credentials give and to revoke tokens linked to such credentials. The loss of a credential linked to a specific device MUST NOT lead to a compromise of other credentials not linked to that device, therefore secret keys used for authentication MUST NOT be shared between more than two parties.

Operators of clients or RS SHOULD have procedures in place to replace credentials that are suspected to have been compromised or that have been lost.

Operators also SHOULD have procedures for decommissioning devices, that include securely erasing credentials and other security critical material in the devices being decommissioned.

6.4. Unprotected AS Request Creation Hints

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the "AS Request Creation Hints" contained in an unprotected response from RS to an unauthorized request (see Section 5.3) are authentic. C therefore MUST determine if an AS is authorized to provide access tokens for a certain RS. How this determination is implemented is out of scope for this document and left to the applications.

6.5. Minimal Security Requirements for Communication

This section summarizes the minimal requirements for the communication security of the different protocol interactions.

C-AS All communication between the client and the Authorization Server MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the client MUST be bound to the client's request to avoid attacks where the attacker swaps the intended response for an older one valid for a previous request. This requires that the client and the Authorization Server have previously exchanged either a shared secret or their public keys in order to negotiate a secure communication. Furthermore the client MUST be able to determine whether an AS has the authority to issue access tokens for a certain RS. This can for example be done through pre-configured lists, or through an online lookup mechanism that in turn also must be secured.

RS-AS The communication between the Resource Server and the Authorization Server via the introspection endpoint MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the RS MUST be bound to the RS's request. This

requires that the RS and the Authorization Server have previously exchanged either a shared secret, or their public keys in order to negotiate a secure communication. Furthermore the RS MUST be able to determine whether an AS has the authority to issue access tokens itself. This is usually configured out of band, but could also be performed through an online lookup mechanism provided that it is also secured in the same way.

C-RS The initial communication between the client and the Resource Server can not be secured in general, since the RS is not in possession of an access token for that client, which would carry the necessary parameters. If both parties support DTLS without client authentication it is RECOMMEND to use this mechanism for protecting the initial communication. After the client has successfully transmitted the access token to the RS, a secure communication protocol MUST be established between client and RS for the actual resource request. This protocol MUST provide confidentiality, integrity and replay protection as well as a binding between requests and responses. This requires that the client learned either the RS's public key or received a symmetric proof-of-possession key bound to the access token from the AS. The RS must have learned either the client's public key or a shared symmetric key from the claims in the token or an introspection request. Since ACE does not provide profile negotiation between C and RS, the client MUST have learned what profile the RS supports (e.g. from the AS or pre-configured) and initiate the communication accordingly.

6.6. Token Freshness and Expiration

An RS that is offline faces the problem of clock drift. Since it cannot synchronize its clock with the AS, it may be tricked into accepting old access tokens that are no longer valid or have been compromised. In order to prevent this, an RS may use the nonce-based mechanism (cnonce) defined in Section 5.3 to ensure freshness of an Access Token subsequently presented to this RS.

Another problem with clock drift is that evaluating the standard token expiration claim "exp" can give unpredictable results.

Acceptable ranges of clock drift are highly dependent on the concrete application. Important factors are how long access tokens are valid, and how critical timely expiration of access token is.

The expiration mechanism implemented by the "exp" claim, based on the first time the RS sees the token was defined to provide a more predictable alternative. The "exp" approach has some drawbacks that need to be considered:

A malicious client may hold back tokens with the "exi" claim in order to prolong their lifespan.

If an RS loses state (e.g. due to an unscheduled reboot), it may lose the current values of counters tracking the "exi" claims of tokens it is storing.

The first drawback is inherent to the deployment scenario and the "exi" solution. It can therefore not be mitigated without requiring the RS be online at times. The second drawback can be mitigated by regularly storing the value of "exi" counters to persistent memory.

6.7. Combining Profiles

There may be use cases where different transport and security protocols are allowed for the different interactions, and, if that is not explicitly covered by an existing profile, it corresponds to combining profiles into a new one. For example, a new profile could specify that a previously-defined MQTT-TLS profile is used between the client and the RS in combination with a previously-defined CoAP-DTLS profile for interactions between the client and the AS. The new profile that combines existing profiles MUST specify how the existing profiles' security properties are achieved. Any profile therefore MUST clearly specify its security requirements and MUST document if its security depends on the combination of various protocol interactions.

6.8. Unprotected Information

Communication with the authz-info endpoint, as well as the various error responses defined in this framework, all potentially include sending information over an unprotected channel. These messages may leak information to an adversary, or may be manipulated by active attackers to induce incorrect behavior. For example error responses for requests to the Authorization Information endpoint can reveal information about an otherwise opaque access token to an adversary who has intercepted this token.

As far as error messages are concerned, this framework is written under the assumption that, in general, the benefits of detailed error messages outweigh the risk due to information leakage. For particular use cases, where this assessment does not apply, detailed error messages can be replaced by more generic ones.

In some scenarios it may be possible to protect the communication with the authz-info endpoint (e.g. through DTLS with only server-side authentication). In cases where this is not possible, it is RECOMMENDED to use encrypted CWTs or tokens that are opaque references and need to be subjected to introspection by the RS.

If the initial unauthorized resource request message (see Section 5.2) is used, the client MUST make sure that it is not sending sensitive content in this request. While GET and DELETE requests only reveal the target URI of the resource, POST and PUT requests would reveal the whole payload of the intended operation.

Since the client is not authenticated at the point when it is submitting an access token to the authz-info endpoint, attackers may be pretending to be a client and trying to trick an RS to use an obsolete profile that in turn specifies a vulnerable security mechanism via the authz-info endpoint. Such an attack would require a valid access token containing an "ace_profile" claim requesting the use of said obsolete profile. Resource Owners should update the configuration of their RS's to prevent them from using such obsolete profiles.

6.9. Identifying Audiences

The audience claim as defined in [RFC7519] and the equivalent "audience" parameter from [RFC8693] are intentionally vague on how to match the audience value to a specific RS. This is intended to allow application specific semantics to be used. This section attempts to give some general guidance for the use of audiences in constrained environments.

URLs are not a good way of identifying mobile devices that can switch networks and thus be associated with new URLs. If the audience represents a single RS, and asymmetric keys are used, the RS can be uniquely identified by a hash of its public key. If this approach is used it is RECOMMENDED to apply the procedure from section 3 of [RFC6920].

If the audience addresses a group of resource servers, the mapping of group identifier to individual RS has to be provisioned to each RS before the group-audience is usable. Managing dynamic groups could be an issue, if any RS is not always reachable when the groups' memberships change. Furthermore, issuing access tokens bound to symmetric proof-of-possession keys that apply to a group-audience is problematic, as an RS that is in possession of the access token can impersonate the client towards the other RSs that are part of the group. It is therefore NOT RECOMMENDED to issue access tokens bound to a group audience and symmetric proof-of possession keys.

Even the client must be able to determine the correct values to put into the "audience" parameter, in order to obtain a token for the intended RS. Errors in this process can lead to the client inadvertently obtaining a token for the wrong RS. The correct values for "audience" can either be provisioned to the client as part of its configuration, or dynamically looked up by the client in some directory. In the latter case the integrity and correctness of the directory data must be assured. Note that the "audience" hint provided by the RS as part of the "AS Request Creation Hints" Section 5.3 is not typically source authenticated and integrity protected, and should therefore not be treated a trusted value.

6.10. Denial of Service Against or with Introspection

The optional introspection mechanism provided by OAuth and supported in the ACE framework allows for two types of attacks that need to be considered by implementers.

First, an attacker could perform a denial of service attack against the introspection endpoint at the AS in order to prevent validation of access tokens. To maintain the security of the system, an RS that is configured to use introspection MUST NOT allow access based on a token for which it couldn't reach the introspection endpoint.

Second, an attacker could use the fact that an RS performs introspection to perform a denial of service attack against that RS by repeatedly sending tokens to its authz-info endpoint that require an introspection call. RS can mitigate such attacks by implementing rate limits on how many introspection requests they perform in a given time interval for a certain client IP address submitting tokens to /authz-info. When that limit has been reached, incoming requests from that address are rejected for a certain amount of time. A general rate limit on the introspection requests should also be considered, to mitigate distributed attacks.

7. Privacy Considerations

Implementers and users should be aware of the privacy implications of the different possible deployments of this framework.

The AS is in a very central position and can potentially learn sensitive information about the clients requesting access tokens. If the client credentials grant is used, the AS can track what kind of access the client intends to perform. With other grants this can be prevented by the Resource Owner. To do so, the resource owner needs to bind the grants it issues to anonymous, ephemeral credentials that do not allow the AS to link different grants and thus different access token requests by the same client.

The claims contained in a token can reveal privacy sensitive information about the client and the RS to any party having access to them (whether by processing the content of a self-contained token or by introspection). The AS SHOULD be configured to minimize the information about clients and RSs disclosed in the tokens it issues.

If tokens are only integrity protected and not encrypted, they may reveal information to attackers listening on the wire, or able to acquire the access tokens in some other way. In the case of CWTs the token may, e.g., reveal the audience, the scope and the confirmation method used by the client. The latter may reveal the identity of the device or application running the client. This may be linkable to the identity of the person using the client (if there is a person and not a machine-to-machine interaction).

Clients using asymmetric keys for proof-of-possession should be aware of the consequences of using the same key pair for proof-of-possession towards different RSs. A set of colluding RSs or an attacker able to obtain the access tokens will be able to link the requests, or even to determine the client's identity.

An unprotected response to an unauthorized request (see Section 5.3) may disclose information about RS and/or its existing relationship with C. It is advisable to include as little information as possible in an unencrypted response. Even the absolute URI of the AS may reveal sensitive information about the service that RS provides. Developers must ensure that the RS does not disclose information that has an impact on the privacy of the stakeholders in the "AS Request Creation Hints". They may choose to use a different mechanism for the discovery of the AS if necessary. If means of encrypting communication between C and RS already exist, more detailed information may be included with an error response to provide C with sufficient information to react on that particular error.

8. IANA Considerations

This document creates several registries with a registration policy of "Expert Review"; guidelines to the experts are given in Section 8.17.

8.1. ACE Authorization Server Request Creation Hints

This specification establishes the IANA "ACE Authorization Server Request Creation Hints" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of the registry are:

Name The name of the parameter

CBOR Key CBOR map key for the parameter. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Value Type The CBOR data types allowable for the values of this parameter.

Reference This contains a pointer to the public specification of the request creation hint abbreviation, if one exists.

This registry will be initially populated by the values in Figure 2. The Reference column for all of these entries will be this document.

8.2. CoRE Resource Type Registry

IANA is requested to register a new Resource Type (rt=) Link Target Attribute in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" [IANA.CoreParameters] registry:

- * Value: ace.ai
- * Description: ACE-OAuth authz-info endpoint resource.
- * Reference: [this document]

Specific ACE-OAuth profiles can use this common resource type for defining their profile-specific discovery processes.

8.3. OAuth Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry [IANA.OAuthExtensionsErrorRegistry].

- * Error name: unsupported_pop_key
- * Error usage location: token error response
- * Related protocol extension: [this document]
- * Change Controller: IETF
- * Specification document(s): Section 5.8.3 of [this document]
- * Error name: incompatible_ace_profiles
- * Error usage location: token error response

- * Related protocol extension: [this document]
- * Change Controller: IETF
- * Specification document(s): Section 5.8.3 of [this document]

8.4. OAuth Error Code CBOR Mappings Registry

This specification establishes the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of the registry are:

Name	The OAuth Error Code name, refers to the name in Section 5.2. of [RFC6749], e.g., "invalid_request".
CBOR Value	CBOR abbreviation for this error code. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the error code abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the error code, if one exists.

This registry will be initially populated by the values in Figure 10. The Reference column for all of these entries will be this document.

8.5. OAuth Grant Type CBOR Mappings

This specification establishes the IANA "OAuth Grant Type CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The name of the grant type as specified in Section 1.3 of [RFC6749].
CBOR Value	CBOR abbreviation for this grant type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the grant type abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the grant type, if one exists.

This registry will be initially populated by the values in Figure 11. The Reference column for all of these entries will be this document.

8.6. OAuth Access Token Types

This section registers the following new token type in the "OAuth Access Token Types" registry [IANA.OAuthAccessTokenTypes].

- * Type name: PoP
- * Additional Token Endpoint Response Parameters: "cnf", "rs_cnf" see section 3.1 of [RFC8747] and section 3.1 of [I-D.ietf-ace-oauth-params].
- * HTTP Authentication Scheme(s): N/A
- * Change Controller: IETF
- * Specification document(s): [this document]

8.7. OAuth Access Token Type CBOR Mappings

This specification established the IANA "OAuth Access Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The name of token type as registered in the OAuth Access Token Types registry, e.g., "Bearer".
CBOR Value	CBOR abbreviation for this token type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the OAuth token type abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the OAuth token type, if one exists.

8.7.1. Initial Registry Contents

- * Name: Bearer
- * Value: 1
- * Reference: [this document]
- * Original Specification: [RFC6749]

- * Name: PoP
- * Value: 2
- * Reference: [this document]
- * Original Specification: [this document]

8.8. ACE Profile Registry

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

The columns of this registry are:

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

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

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

Reference This contains a pointer to the public specification of the profile abbreviation, if one exists.

This registry will be initially empty and will be populated by the registrations from the ACE framework profiles.

8.9. OAuth Parameter Registration

This specification registers the following parameter in the "OAuth Parameters" registry [IANA.OAuthParameters]:

- * Name: ace_profile
- * Parameter Usage Location: token response
- * Change Controller: IETF
- * Reference: Section 5.8.2 and Section 5.8.4.3 of [this document]

8.10. OAuth Parameters CBOR Mappings Registry

This specification establishes the IANA "OAuth Parameters CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".

CBOR Key CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].

Value Type The allowable CBOR data types for values of this parameter.

Reference This contains a pointer to the public specification of the OAuth parameter abbreviation, if one exists.

Original Specification This contains a pointer to the public specification of the OAuth parameter, if one exists.

This registry will be initially populated by the values in Figure 12. The Reference column for all of these entries will be this document.

8.11. OAuth Introspection Response Parameter Registration

This specification registers the following parameters in the OAuth Token Introspection Response registry [IANA.TokenIntrospectionResponse].

- * **Name:** ace_profile
- * **Description:** The ACE profile used between client and RS.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** cnonce
- * **Description:** "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** cti
- * **Description:** "CWT ID". The identifier of a CWT as defined in [RFC8392].
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** exp
- * **Description:** "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker form of token expiration for devices that cannot synchronize their internal clocks.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

8.12. OAuth Token Introspection Response CBOR Mappings Registry

This specification establishes the IANA "OAuth Token Introspection Response CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".
CBOR Key	CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Value Type	The allowable CBOR data types for values of this parameter.
Reference	This contains a pointer to the public specification of the introspection response parameter abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of OAuth Token Introspection parameter, if one exists.

This registry will be initially populated by the values in Figure 16. The Reference column for all of these entries will be this document.

Note that the mappings of parameters corresponding to claim names intentionally coincide with the CWT claim name mappings from [RFC8392].

8.13. JSON Web Token Claims

This specification registers the following new claims in the JSON Web Token (JWT) registry of JSON Web Token Claims [IANA.JsonWebTokenClaims]:

- * Claim Name: ace_profile
- * Claim Description: The ACE profile a token is supposed to be used with.
- * Change Controller: IETF
- * Reference: Section 5.10 of [this document]

- * Claim Name: cnonce
- * Claim Description: "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- * Change Controller: IETF
- * Reference: Section 5.10 of [this document]

- * Claim Name: `exp`
- * Claim Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker form of token expiration for devices that cannot synchronize their internal clocks.
- * Change Controller: IETF
- * Reference: Section 5.10.3 of [this document]

8.14. CBOR Web Token Claims

This specification registers the following new claims in the "CBOR Web Token (CWT) Claims" registry [IANA.CborWebTokenClaims].

- * Claim Name: `ace_profile`
- * Claim Description: The ACE profile a token is supposed to be used with.
- * JWT Claim Name: `ace_profile`
- * Claim Key: TBD (suggested: 38)
- * Claim Value Type(s): integer
- * Change Controller: IETF
- * Specification Document(s): Section 5.10 of [this document]

- * Claim Name: `cnonce`
- * Claim Description: The client-nonce sent to the AS by the RS via the client.
- * JWT Claim Name: `cnonce`
- * Claim Key: TBD (suggested: 39)
- * Claim Value Type(s): byte string
- * Change Controller: IETF
- * Specification Document(s): Section 5.10 of [this document]

- * Claim Name: `exp`
- * Claim Description: The expiration time of a token measured from when it was received at the RS in seconds.
- * JWT Claim Name: `exp`
- * Claim Key: TBD (suggested: 40)
- * Claim Value Type(s): integer
- * Change Controller: IETF
- * Specification Document(s): Section 5.10.3 of [this document]

- * Claim Name: `scope`
- * Claim Description: The scope of an access token as defined in [RFC6749].
- * JWT Claim Name: `scope`
- * Claim Key: TBD (suggested: 9)
- * Claim Value Type(s): byte string or text string
- * Change Controller: IETF
- * Specification Document(s): Section 4.2 of [RFC8693]

8.15. Media Type Registrations

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

Type name: application

Subtype name: ace+cbor

Required parameters: N/A

Optional parameters: N/A

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

Security considerations: See Section 6 of [this document]

Interoperability considerations: N/A

Published specification: [this document]

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

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information:
<iesg@ietf.org>

Intended usage: COMMON

Restrictions on usage: none

Author: Ludwig Seitz <ludwig.seitz@combitech.se>

Change controller: IETF

8.16. CoAP Content-Format Registry

This specification registers the following entry to the "CoAP Content-Formats" registry:

Media Type: application/ace+cbor

Encoding: -

ID: TBD (suggested: 19)

Reference: [this document]

8.17. Expert Review Instructions

All of the IANA registries established in this document are defined to use a registration policy of 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 needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- * Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on.
- * Since a high degree of overlap is expected between these registries and the contents of the OAuth parameters [IANA.OAuthParameters] registries, experts should require new registrations to maintain alignment with parameters from OAuth that have comparable functionality. Deviation from this alignment should only be allowed if there are functional differences, that are motivated by the use case and that cannot be easily or efficiently addressed by comparable OAuth parameters.

9. Acknowledgments

This document is a product of the ACE working group of the IETF.

Thanks to Eve Maler for her contributions to the use of OAuth 2.0 and UMA in IoT scenarios, Robert Taylor for his discussion input, and Malisa Vucinic for his input on the predecessors of this proposal.

Thanks to the authors of draft-ietf-oauth-pop-key-distribution, from where parts of the security considerations were copied.

Thanks to Stefanie Gerdes, Olaf Bergmann, and Carsten Bormann for contributing their work on AS discovery from draft-gerdes-ace-dcaf-authorize (see Section 5.1) and the considerations on multiple access tokens.

Thanks to Jim Schaad and Mike Jones for their comprehensive reviews.

Thanks to Benjamin Kaduk for his input on various questions related to this work.

Thanks to Cigdem Sengul for some very useful review comments.

Thanks to Carsten Bormann for contributing the text for the CoRE Resource Type registry.

Thanks to Roman Danyliw for suggesting the Appendix E (including its contents).

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova. Ludwig Seitz was also received further funding for this work by Vinnova in the context of the CelticNext project Critisec.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-params]

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-16, 7 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-16.txt>>.

- [IANA.CborWebTokenClaims]
IANA, "CBOR Web Token (CWT) Claims",
<<https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>>.
- [IANA.CoreParameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>>.
- [IANA.JsonWebTokenClaims]
IANA, "JSON Web Token Claims",
<<https://www.iana.org/assignments/jwt/jwt.xhtml#claims>>.
- [IANA.OAuthAccessTokenTypes]
IANA, "OAuth Access Token Types",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-types>>.
- [IANA.OAuthExtensionsErrorRegistry]
IANA, "OAuth Extensions Error Registry",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#extensions-error>>.
- [IANA.OAuthParameters]
IANA, "OAuth Parameters",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#parameters>>.
- [IANA.TokenIntrospectionResponse]
IANA, "OAuth Token Introspection Response",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-introspection-response>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [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>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<https://www.rfc-editor.org/info/rfc6750>>.
- [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>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/info/rfc8693>>.
- [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>>.

10.2. Informative References

- [BLE] Bluetooth SIG, "Bluetooth Core Specification v5.1", Section 4.4, January 2019, <<https://www.bluetooth.com/specifications/bluetooth-core-specification/>>.
- [I-D.erdtdman-ace-rpcc]
Seitz, L. and S. Erdtman, "Raw-Public-Key and Pre-Shared-Key as OAuth client credentials", Work in Progress, Internet-Draft, draft-erdtdman-ace-rpcc-02, 30 October 2017, <<https://www.ietf.org/archive/id/draft-erdtdman-ace-rpcc-02.txt>>.
- [I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

[I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-quic-transport-34.txt>>.

[I-D.ietf-tls-dtls13]

Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.

[Margil0impact]

Margi, C. B., de Oliveira, B.T., de Sousa, G.T., Simplicio Jr, M.A., Barreto, P.S.L.M., Carvalho, T.C.M.B., Naeslund, M., and R. Gold, "Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds", Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), August 2010.

[MQTT5.0] Banks, A., Briggs, E., Borgendale, K., and R. Gupta, "MQTT Version 5.0", OASIS Standard, March 2019, <<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>>.

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

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<https://www.rfc-editor.org/info/rfc6819>>.

- [RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <<https://www.rfc-editor.org/info/rfc7009>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <<https://www.rfc-editor.org/info/rfc7521>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/info/rfc7591>>.
- [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>>.
- [RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <<https://www.rfc-editor.org/info/rfc7744>>.
- [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>>.
- [RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <<https://www.rfc-editor.org/info/rfc8252>>.

- [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>>.
- [RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <<https://www.rfc-editor.org/info/rfc8414>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8516] Keranen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.
- [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>>.
- [RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <<https://www.rfc-editor.org/info/rfc8628>>.

Appendix A. Design Justification

This section provides further insight into the design decisions of the solution documented in this document. Section 3 lists several building blocks and briefly summarizes their importance. The justification for offering some of those building blocks, as opposed to using OAuth 2.0 as is, is given below.

Common IoT constraints are:

Low Power Radio:

Many IoT devices are equipped with a small battery which needs to last for a long time. For many constrained wireless devices, the highest energy cost is associated to transmitting or receiving messages (roughly by a factor of 10 compared to AES) [Margil10impact]. It is therefore important to keep the total communication overhead low, including minimizing the number and size of messages sent and received, which has an impact of choice on the message format and protocol. By using CoAP over UDP and

CBOR encoded messages, some of these aspects are addressed. Security protocols contribute to the communication overhead and can, in some cases, be optimized. For example, authentication and key establishment may, in certain cases where security requirements allow, be replaced by provisioning of security context by a trusted third party, using transport or application-layer security.

Low CPU Speed:

Some IoT devices are equipped with processors that are significantly slower than those found in most current devices on the Internet. This typically has implications on what timely cryptographic operations a device is capable of performing, which in turn impacts, e.g., protocol latency. Symmetric key cryptography may be used instead of the computationally more expensive public key cryptography where the security requirements so allow, but this may also require support for trusted-third-party-assisted secret key establishment using transport- or application-layer security.

Small Amount of Memory:

Microcontrollers embedded in IoT devices are often equipped with only a small amount of RAM and flash memory, which places limitations on what kind of processing can be performed and how much code can be put on those devices. To reduce code size, fewer and smaller protocol implementations can be put on the firmware of such a device. In this case, CoAP may be used instead of HTTP, symmetric-key cryptography instead of public-key cryptography, and CBOR instead of JSON. An authentication and key establishment protocol, e.g., the DTLS handshake, in comparison with assisted key establishment, also has an impact on memory and code footprints.

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers may not want to give access to the data collected by their IoT device or to functions it may offer to third parties. Since the classical approach of requesting permissions from end users via a rich user interface does not work in many IoT deployment scenarios, these functions need to be delegated to user-controlled devices that are better suitable for such tasks, such as smart phones and tablets.

Communication Constraints:

In certain constrained settings an IoT device may not be able to communicate with a given device at all times. Devices may be sleeping, or just disconnected from the Internet because of general lack of connectivity in the area, for cost reasons, or for security reasons, e.g., to avoid an entry point for Denial-of-Service attacks.

The communication interactions this framework builds upon (as shown graphically in Figure 1) may be accomplished using a variety of different protocols, and not all parts of the message flow are used in all applications due to the communication constraints. Deployments making use of CoAP are expected, but this framework is not limited to them. Other protocols such as HTTP, or even protocols such as Bluetooth Smart communication that do not necessarily use IP, could also be used. The latter raises the need for application-layer security over the various interfaces.

In the light of these constraints we have made the following design decisions:

CBOR, COSE, CWT:

When using this framework, it is RECOMMENDED to use CBOR [RFC8949] as data format. Where CBOR data needs to be protected, the use of COSE [RFC8152] is RECOMMENDED. Furthermore, where self-contained tokens are needed, it is RECOMMENDED to use of CWT [RFC8392]. These measures aim at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

CoAP:

When using this framework, it is RECOMMENDED to use of CoAP [RFC7252] instead of HTTP. This does not preclude the use of other protocols specifically aimed at constrained devices, like, e.g., Bluetooth Low Energy (see Section 3.2). This aims again at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

Access Information:

This framework defines the name "Access Information" for data concerning the RS that the AS returns to the client in an access token response (see Section 5.8.2). This aims at enabling scenarios where a powerful client, supporting multiple profiles, needs to interact with an RS for which it does not know the supported profiles and the raw public key.

Proof-of-Possession:

This framework makes use of proof-of-possession tokens, using the "cnf" claim [RFC8747]. A request parameter "cnf" and a Response parameter "cnf", both having a value space semantically and syntactically identical to the "cnf" claim, are defined for the token endpoint, to allow requesting and stating confirmation keys. This aims at making token theft harder. Token theft is specifically relevant in constrained use cases, as communication often passes through middle-boxes, which could be able to steal bearer tokens and use them to gain unauthorized access.

Authz-Info endpoint:

This framework introduces a new way of providing access tokens to an RS by exposing a authz-info endpoint, to which access tokens can be POSTed. This aims at reducing the size of the request message and the code complexity at the RS. The size of the request message is problematic, since many constrained protocols have severe message size limitations at the physical layer (e.g., in the order of 100 bytes). This means that larger packets get fragmented, which in turn combines badly with the high rate of packet loss, and the need to retransmit the whole message if one packet gets lost. Thus separating sending of the request and sending of the access tokens helps to reduce fragmentation.

Client Credentials Grant:

In this framework the use of the client credentials grant is RECOMMENDED for machine-to-machine communication use cases, where manual intervention of the resource owner to produce a grant token is not feasible. The intention is that the resource owner would instead pre-arrange authorization with the AS, based on the client's own credentials. The client can then (without manual intervention) obtain access tokens from the AS.

Introspection:

In this framework the use of access token introspection is RECOMMENDED in cases where the client is constrained in a way that it can not easily obtain new access tokens (i.e. it has connectivity issues that prevent it from communicating with the AS). In that case it is RECOMMENDED to use a long-term token, that could be a simple reference. The RS is assumed to be able to communicate with the AS, and can therefore perform introspection, in order to learn the claims associated with the token reference. The advantage of such an approach is that the resource owner can change the claims associated to the token reference without having to be in contact with the client, thus granting or revoking access rights.

Appendix B. Roles and Responsibilities

Resource Owner

- * Make sure that the RS is registered at the AS. This includes making known to the AS which profiles, token_type, scopes, and key types (symmetric/asymmetric) the RS supports. Also making it known to the AS which audience(s) the RS identifies itself with.
- * Make sure that clients can discover the AS that is in charge of the RS.
- * If the client-credentials grant is used, make sure that the AS has the necessary, up-to-date, access control policies for the RS.

Requesting Party

- * Make sure that the client is provisioned the necessary credentials to authenticate to the AS.
- * Make sure that the client is configured to follow the security requirements of the Requesting Party when issuing requests (e.g., minimum communication security requirements, trust anchors).
- * Register the client at the AS. This includes making known to the AS which profiles, token_types, and key types (symmetric/asymmetric) the client.

Authorization Server

- * Register the RS and manage corresponding security contexts.
- * Register clients and authentication credentials.
- * Allow Resource Owners to configure and update access control policies related to their registered RSs.
- * Expose the token endpoint to allow clients to request tokens.
- * Authenticate clients that wish to request a token.
- * Process a token request using the authorization policies configured for the RS.
- * Optionally: Expose the introspection endpoint that allows RS's to submit token introspection requests.
- * If providing an introspection endpoint: Authenticate RSs that wish to get an introspection response.
- * If providing an introspection endpoint: Process token introspection requests.
- * Optionally: Handle token revocation.
- * Optionally: Provide discovery metadata. See [RFC8414]
- * Optionally: Handle refresh tokens.

Client

- * Discover the AS in charge of the RS that is to be targeted with a request.
- * Submit the token request (see step (A) of Figure 1).
 - Authenticate to the AS.
 - Optionally (if not pre-configured): Specify which RS, which resource(s), and which action(s) the request(s) will target.
 - If raw public keys (rpk) or certificates are used, make sure the AS has the right rpk or certificate for this client.
- * Process the access token and Access Information (see step (B) of Figure 1).
 - Check that the Access Information provides the necessary security parameters (e.g., PoP key, information on communication security protocols supported by the RS).
 - Safely store the proof-of-possession key.
 - If provided by the AS: Safely store the refresh token.
- * Send the token and request to the RS (see step (C) of Figure 1).
 - Authenticate towards the RS (this could coincide with the proof of possession process).

- Transmit the token as specified by the AS (default is to the authz-info endpoint, alternative options are specified by profiles).
 - Perform the proof-of-possession procedure as specified by the profile in use (this may already have been taken care of through the authentication procedure).
 - * Process the RS response (see step (F) of Figure 1) of the RS.
- Resource Server
- * Expose a way to submit access tokens. By default this is the authz-info endpoint.
 - * Process an access token.
 - Verify the token is from a recognized AS.
 - Check the token's integrity.
 - Verify that the token applies to this RS.
 - Check that the token has not expired (if the token provides expiration information).
 - Store the token so that it can be retrieved in the context of a matching request.
- Note: The order proposed here is not normative, any process that arrives at an equivalent result can be used. A noteworthy consideration is whether one can use cheap operations early on to quickly discard non-applicable or invalid tokens, before performing expensive cryptographic operations (e.g. doing an expiration check before verifying a signature).
- * Process a request.
 - Set up communication security with the client.
 - Authenticate the client.
 - Match the client against existing tokens.
 - Check that tokens belonging to the client actually authorize the requested action.
 - Optionally: Check that the matching tokens are still valid, using introspection (if this is possible.)
 - * Send a response following the agreed upon communication security mechanism(s).
 - * Safely store credentials such as raw public keys for authentication or proof-of-possession keys linked to access tokens.

Appendix C. Requirements on Profiles

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

- * Optionally define new methods for the client to discover the necessary permissions and AS for accessing a resource, different from the one proposed in Section 5.1. Section 4
- * Optionally specify new grant types. Section 5.4

- * Optionally define the use of client certificates as client credential type. Section 5.5
- * Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.8.4.3
- * Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS). This must provide encryption, integrity and replay protection. Section 5.8.4.3
- * Specify how the client and the RS mutually authenticate. Section 4
- * Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol. Section 5.8.4.2
- * Specify a unique `ace_profile` identifier. Section 5.8.4.3
- * If introspection is supported: Specify the communication and security protocol for introspection. Section 5.9
- * Specify the communication and security protocol for interactions between client and AS. This must provide encryption, integrity protection, replay protection and a binding between requests and responses. Section 5 and Section 5.8
- * Specify how/if the `authz-info` endpoint is protected, including how error responses are protected. Section 5.10.1
- * Optionally define other methods of token transport than the `authz-info` endpoint. Section 5.10.1

Appendix D. Assumptions on AS Knowledge about C and RS

This section lists the assumptions on what an AS should know about a client and an RS in order to be able to respond to requests to the token and introspection endpoints. How this information is established is out of scope for this document.

- * The identifier of the client or RS.
- * The profiles that the client or RS supports.
- * The scopes that the RS supports.
- * The audiences that the RS identifies with.
- * The key types (e.g., pre-shared symmetric key, raw public key, key length, other key parameters) that the client or RS supports.
- * The types of access tokens the RS supports (e.g., CWT).
- * If the RS supports CWTs, the COSE parameters for the crypto wrapper (e.g., algorithm, key-wrap algorithm, key-length) that the RS supports.
- * The expiration time for access tokens issued to this RS (unless the RS accepts a default time chosen by the AS).
- * The symmetric key shared between client and AS (if any).
- * The symmetric key shared between RS and AS (if any).
- * The raw public key of the client or RS (if any).

- * Whether the RS has synchronized time (and thus is able to use the 'exp' claim) or not.

Appendix E. Differences to OAuth 2.0

This document adapts OAuth 2.0 to be suitable for constrained environments. This section lists the main differences from the normative requirements of OAuth 2.0.

- * Use of TLS -- OAuth 2.0 requires the use of TLS both to protect the communication between AS and client when requesting an access token; between client and RS when accessing a resource and between AS and RS if introspection is used. This framework requires similar security properties, but does not require that they be realized with TLS. See Section 5.
- * Cardinality of "grant_type" parameter -- In client-to-AS requests using OAuth 2.0, the "grant_type" parameter is required (per [RFC6749]). In this framework, this parameter is optional. See Section 5.8.1.
- * Encoding of "scope" parameter -- In client-to-AS requests using OAuth 2.0, the "scope" parameter is string encoded (per [RFC6749]). In this framework, this parameter may also be encoded as a byte string. See Section 5.8.1.
- * Cardinality of "token_type" parameter -- in AS-to-client responses using OAuth 2.0, the token_type parameter is required (per [RFC6749]). In this framework, this parameter is optional. See Section 5.8.2.
- * Access token retention -- in OAuth 2.0, the access token may be sent with every request to the RS. The exact use of access tokens depends on the semantics of the application and the session management concept it uses. In this framework, the RS must be able to store these tokens for later use. See Section 5.10.1.

Appendix F. Deployment Examples

There is a large variety of IoT deployments, as is indicated in Appendix A, and this section highlights a few common variants. This section is not normative but illustrates how the framework can be applied.

For each of the deployment variants, there are a number of possible security setups between clients, resource servers and authorization servers. The main focus in the following subsections is on how authorization of a client request for a resource hosted by an RS is performed. This requires the security of the requests and responses between the clients and the RS to be considered.

Note: CBOR diagnostic notation is used for examples of requests and responses.

F.1. Local Token Validation

In this scenario, the case where the resource server is offline is considered, i.e., it is not connected to the AS at the time of the access request. This access procedure involves steps A, B, C, and F of Figure 1.

Since the resource server must be able to verify the access token locally, self-contained access tokens must be used.

This example shows the interactions between a client, the authorization server and a temperature sensor acting as a resource server. Message exchanges A and B are shown in Figure 17.

A: The client first generates a public-private key pair used for communication security with the RS. The client sends a CoAP POST request to the token endpoint at the AS. The security of this request can be transport or application layer. It is up to the communication security profile to define. In the example it is assumed that both client and AS have performed mutual authentication e.g. via DTLS. The request contains the public key of the client and the Audience parameter set to "tempSensorInLivingRoom", a value that the temperature sensor identifies itself with. The AS evaluates the request and authorizes the client to access the resource.

B: The AS responds with a 2.05 Content response containing the Access Information, including the access token. The PoP access token contains the public key of the client, and the Access Information contains the public key of the RS. For communication security this example uses DTLS RawPublicKey between the client and the RS. The issued token will have a short validity time, i.e., "exp" close to "iat", in order to mitigate attacks using stolen client credentials. The token includes the claim such as "scope" with the authorized access that an owner of the temperature device can enjoy. In this example, the "scope" claim, issued by the AS, informs the RS that the owner of the token, that can prove the possession of a key is authorized to make a GET request against the /temperature resource and a POST request on the /firmware resource. Note that the syntax and semantics of the scope claim are application specific.

Note: In this example it is assumed that the client knows what resource it wants to access, and is therefore able to request specific audience and scope claims for the access token.

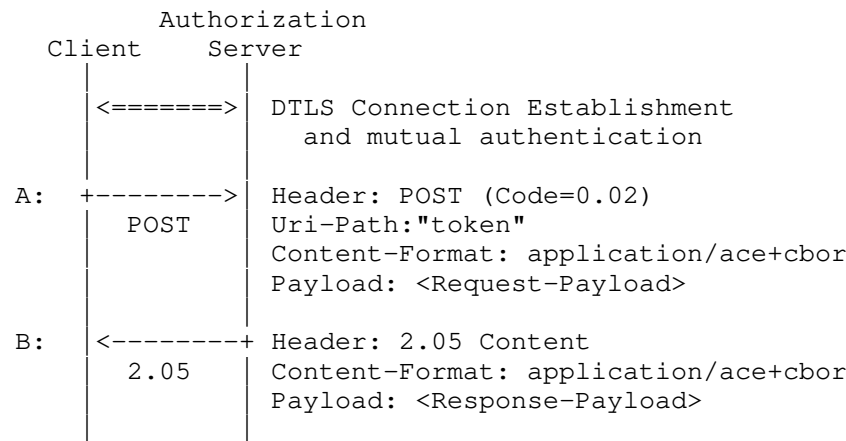


Figure 17: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 18 Note that the parameter "rs_cnf" from [I-D.ietf-ace-oauth-params] is used to inform the client about the resource server's public key.

```

Request-Payload :
{
  "audience" : "tempSensorInLivingRoom",
  "client_id" : "myclient",
  "req_cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

Response-Payload :
{
  "access_token" : b64'0INDoQEkoQVnKkXfb7xaWqMTf6 ...',
  "rs_cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'MKBCTNIcKUSDii1lySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y" : b64'4Et16SRW2YiLUrN5vfvVHuhp7x8Px1tmWWlbbM4IFyM'
    }
  }
}

```

Figure 18: Request and Response Payload Details.

The content of the access token is shown in Figure 19.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1563451500",
  "exp" : "1563453000",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

```

Figure 19: Access Token including Public Key of the client.

Messages C and F are shown in Figure 20 - Figure 21.

C: The client then sends the PoP access token to the authz-info endpoint at the RS. This is a plain CoAP POST request, i.e., no transport or application-layer security is used between client and RS since the token is integrity protected between the AS and RS. The RS verifies that the PoP access token was created by a known and trusted AS, that it applies to this RS, and that it is valid. The RS caches the security context together with authorization information about this client contained in the PoP access token.

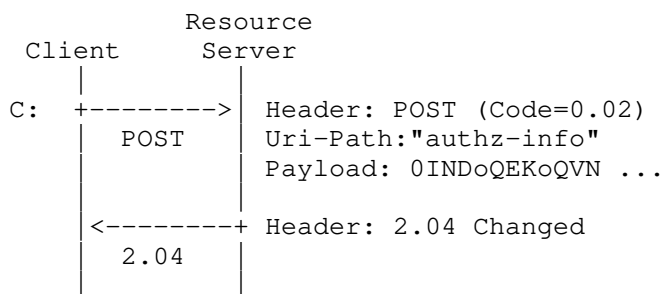


Figure 20: Access Token provisioning to RS

The client and the RS runs the DTLS handshake using the raw public keys established in step B and C.

The client sends a CoAP GET request to /temperature on RS over DTLS. The RS verifies that the request is authorized, based on previously established security context.

F: The RS responds over the same DTLS channel with a CoAP 2.05 Content response, containing a resource representation as payload.

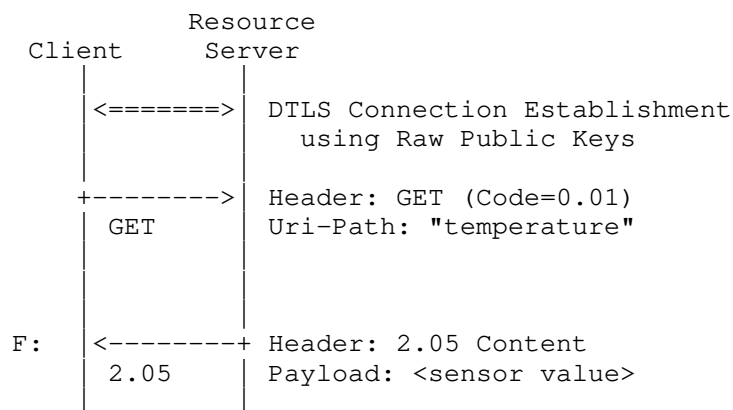


Figure 21: Resource Request and Response protected by DTLS.

F.2. Introspection Aided Token Validation

In this deployment scenario it is assumed that a client is not able to access the AS at the time of the access request, whereas the RS is assumed to be connected to the back-end infrastructure. Thus the RS can make use of token introspection. This access procedure involves steps A-F of Figure 1, but assumes steps A and B have been carried out during a phase when the client had connectivity to AS.

Since the client is assumed to be offline, at least for a certain period of time, a pre-provisioned access token has to be long-lived. Since the client is constrained, the token will not be self contained (i.e. not a CWT) but instead just a reference. The resource server uses its connectivity to learn about the claims associated to the access token by using introspection, which is shown in the example below.

In the example interactions between an offline client (key fob), an RS (online lock), and an AS is shown. It is assumed that there is a provisioning step where the client has access to the AS. This corresponds to message exchanges A and B which are shown in Figure 22.

Authorization consent from the resource owner can be pre-configured, but it can also be provided via an interactive flow with the resource owner. An example of this for the key fob case could be that the resource owner has a connected car, he buys a generic key that he wants to use with the car. To authorize the key fob he connects it to his computer that then provides the UI for the device. After that OAuth 2.0 implicit flow can be used to authorize the key for his car at the car manufacturer's AS.

Note: In this example the client does not know the exact door it will be used to access since the token request is not sent at the time of access. So the scope and audience parameters are set quite wide to start with, while tailored values narrowing down the claims to the specific RS being accessed can be provided to that RS during an introspection step.

A: The client sends a CoAP POST request to the token endpoint at AS. The request contains the Audience parameter set to "PACS1337" (PACS, Physical Access System), a value that identifies the physical access control system to which the individual doors are connected. The AS generates an access token as an opaque string, which it can match to the specific client and the targeted audience. It furthermore generates a symmetric proof-of-

possession key. The communication security and authentication between client and AS is assumed to have been provided at transport layer (e.g. via DTLS) using a pre-shared security context (psk, rpk or certificate).

B: The AS responds with a CoAP 2.05 Content response, containing as payload the Access Information, including the access token and the symmetric proof-of-possession key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key is used as the PreSharedKey.

Note: In this example we are using a symmetric key for a multi-RS audience, which is not recommended normally (see Section 6.9). However in this case the risk is deemed to be acceptable, since all the doors are part of the same physical access control system, and therefore the risk of a malicious RS impersonating the client towards another RS is low.

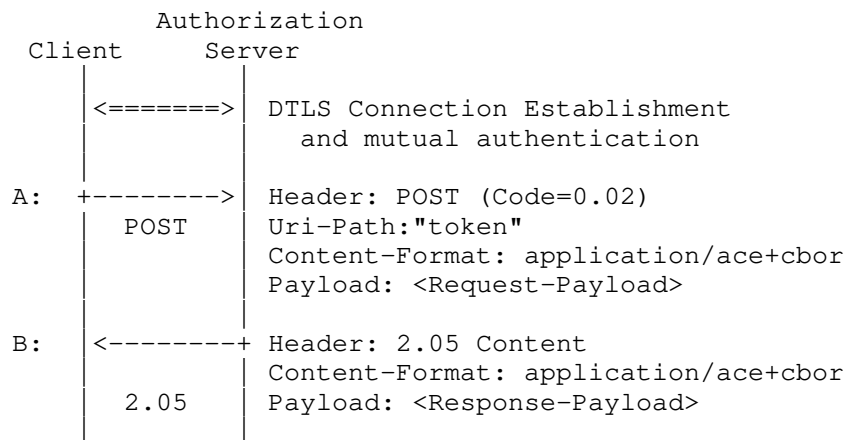


Figure 22: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 23.


```

Request-Payload:
{
  "client_id" : "keyfob",
  "audience" : "PACS1337"
}

Response-Payload:
{
  "access_token" : b64'VGVzdCB0b2t1bG==',
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "oct",
      "alg" : "HS256",
      "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
    }
  }
}

```

Figure 23: Request and Response Payload for C offline

The access token in this case is just an opaque byte string referencing the authorization information at the AS.

C: Next, the client POSTs the access token to the authz-info endpoint in the RS. This is a plain CoAP request, i.e., no DTLS between client and RS. Since the token is an opaque string, the RS cannot verify it on its own, and thus defers to respond the client with a status code until after step E.

D: The RS sends the token to the introspection endpoint on the AS using a CoAP POST request. In this example RS and AS are assumed to have performed mutual authentication using a pre shared security context (psk, rpki or certificate) with the RS acting as DTLS client.

E: The AS provides the introspection response (2.05 Content) containing parameters about the token. This includes the confirmation key (cnf) parameter that allows the RS to verify the client's proof of possession in step F. Note that our example in Figure 25 assumes a pre-established key (e.g. one used by the client and the RS for a previous token) that is now only referenced by its key-identifier 'kid'.

After receiving message E, the RS responds to the client's POST in step C with the CoAP response code 2.01 (Created).

```

Resource
Client  Server
|       |
C: +----->| Header: POST (T=CON, Code=0.02)

```

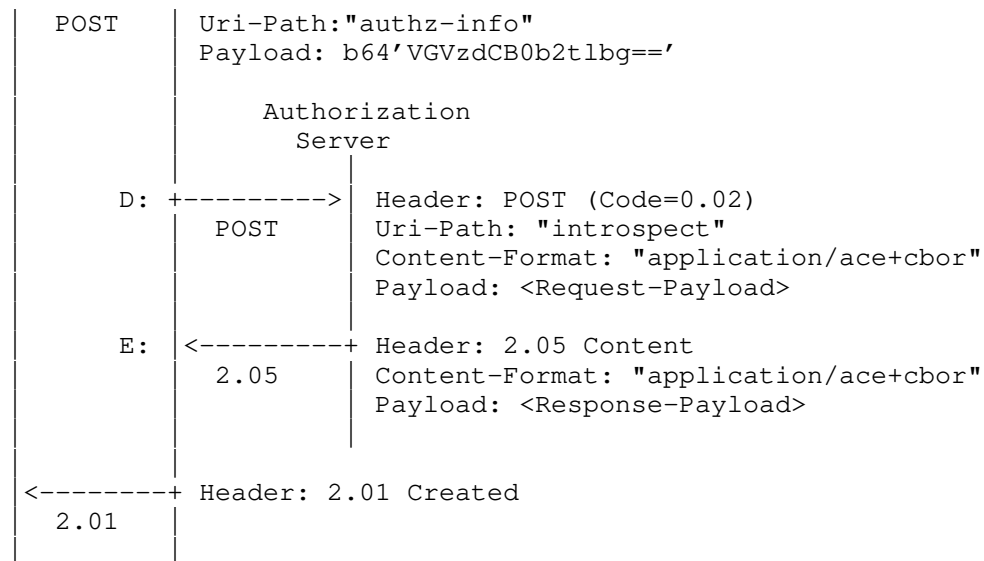


Figure 24: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 25.

Request-Payload:

```

{
  "token" : b64'VGZzdCB0b2t1bg==' ,
  "client_id" : "FrontDoor",
}

```

Response-Payload:

```

{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1563454000,
  "cnf" : {
    "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk'
  }
}

```

Figure 25: Request and Response Payload for Introspection

The client uses the symmetric PoP key to establish a DTLS PreSharedKey secure connection to the RS. The CoAP request PUT is sent to the uri-path /state on the RS, changing the state of the door to locked.

F: The RS responds with a appropriate over the secure DTLS channel.

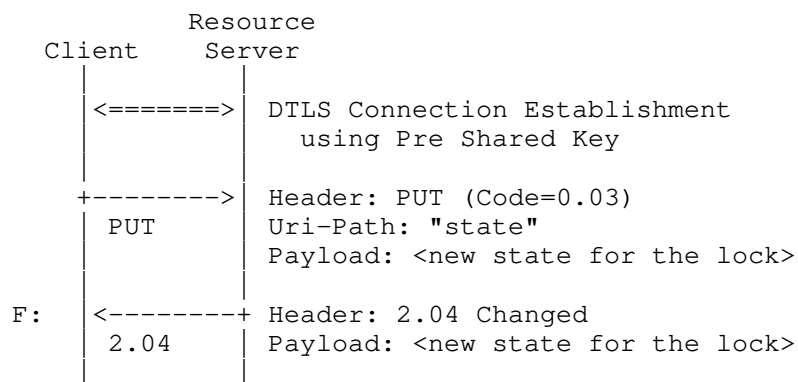


Figure 26: Resource request and response protected by OSCORE

Authors' Addresses

Ludwig Seitz
 Combitech
 Djäknegatan 31
 SE-211 35 Malmö
 Sweden

Email: ludwig.seitz@combitech.com

Goeran Selander
 Ericsson
 Faroegatan 6
 SE-164 80 Kista
 Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
 Sweden

Email: erik@wahlstromstekniska.se

Samuel Erdtman
 Spotify AB
 Birger Jarlsgatan 61, 4tr
 SE-113 56 Stockholm
 Sweden

Email: erdtman@spotify.com

Hannes Tschofenig
Arm Ltd.
6067 Absam
Austria

Email: Hannes.Tschofenig@arm.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 9, 2019

S. Echeverria
CMU SEI
L. Seitz
RISE
D. Klinedinst
G. Lewis
CMU SEI
March 8, 2019

ACE Clients in Disadvantaged Networks
draft-secheverria-ace-client-disadvantaged-00

Abstract

This document describes a set of recommendations to use when implementing ACE/OAuth 2.0 clients that are working in disadvantaged networks. Issues such as token revocation have a much higher priority in scenarios where Resource Servers are IoT devices, and network connectivity is limited and intermittent.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 9, 2019.

Copyright Notice

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

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Sample Scenario	3
3. Recommendations	3
3.1. Use of Client Introspection for Token Revocation	3
3.1.1. Procedure	3
3.1.2. Specific Recommendations	4
3.1.3. Alternatives	5
4. IANA Considerations	6
5. Security Considerations	6
6. Acknowledgements	6
7. Normative References	6
Authors' Addresses	7

1. Introduction

Authentication and authorization in IoT (Internet of Things) devices can be difficult due to constraints in terms of memory, processing, user interface, power and communication bandwidth. OAuth 2.0 and derived standards, such as ACE, can be still applied to these scenarios, often with some modifications. However, when IoT devices are working in disadvantaged networks, there are even greater constraints in terms of communication bandwidth. Nodes in disadvantaged networks operate in what are called DIL environments (disconnected, intermittent, limited), which means that there is limited and unreliable connectivity between nodes with potentially periods of full disconnection. This document will focus on practices that are recommended for clients using ACE/OAuth 2.0 while working with IoT devices in disadvantaged networks.

There are cases in which a client may need to obtain further information about a token without communicating with a Resource Server (RS). One such case is when a client needs to know the active status of a token that it possesses. This is particularly useful in disadvantaged environments where RS impersonation and sabotage are likely threats.

Section 2 describes a sample scenario and Section 3 describes recommendations for client implementation, including the use of client introspection: ensuring only authorized clients can perform client introspection, enabling decryption of self-contained tokens, and limiting information returned in the introspection response.

2. Sample Scenario

A sample scenario is the following: let's assume we have IoT devices that are deployed over a large area to monitor it after an earthquake. These IoT devices may be small sensors of different types (temperature, motion detectors, etc.) that are constantly collecting information from their environment. Each of these IoT devices will act as an RS, and we want to be able to give authorization to access their resources to mobile clients. The Authorization Server (AS) will be mostly static, or slow moving; it could be deployed in a nearby building, or carried along in vehicles if there is no central location. Clients would most likely be smartphones or tablets, carried by users in the field. Due to the mobility of the clients and the large area over which the RSs are deployed, clients would only intermittently have connectivity to both the AS and to each RS. Clients would ask the AS for access tokens when they are in range of the AS, and use the tokens to get information from the RSs when they are in range of the IoT devices.

In this situation, being opportunistic about what to do when a client gets in range of an AS is an important thing to consider. It is also highly likely that clients or RSs may be impersonated or sabotaged. This makes it a high priority to identify tokens associated to a compromised RS or sent to a compromised client.

A specific situation for this would be if the AS admin learns that a certain RS has been compromised. The AS does not have constant connectivity to clients, so it can't let them know right away about the issue. However, it wants to prevent all clients that had tokens to communicate with that RS that they should no longer use those tokens. The AS admin can manually mark all tokens issued to that RS as an audience as revoked (internally). However, a means to let clients know about the revocation of their tokens would be needed.

3. Recommendations

3.1. Use of Client Introspection for Token Revocation

3.1.1. Procedure

One way to let clients know when a token has been revoked is to extend the existing protocol to add specific messages to handle this. But an alternative, simpler way would be to use client introspection. The end goal is to be able to revoke tokens for a RS that has been compromised, by letting clients poll information about the tokens, and then letting them know that they have been revoked.

A client can opportunistically poll an AS using the same introspection mechanism defined in the OAuth 2.0 Token Introspection RFC [RFC7662], to obtain information on whether a specific token is still valid or not. That RFC defines a method for querying an Authorization Server (AS) for metadata about a token. The introspection process focuses on how a Resource Server (RS) could benefit from this information. This is because, in most cases, the token is assumed to be opaque to the client, as it is intended to be a secure way of sending information to a RS, without the client being able to modify it.

This same mechanism can be used to detect revoked tokens opportunistically whenever a client gets in range of an AS. It should work in the following way:

1- A client that gets in range of the AS, uses that opportunity to contact the AS and to ask it about the state of its non-expired tokens. More specifically, it sends a client introspection request for each of the non-expired tokens it is using.

2- The AS replies to the client with information about whether each token is "active" or not. For every token that has been revoked, it returns that the token is not active.

3- The client receives the response and purges non-active tokens from its list of tokens.

Thus the client will be protected from contacting the compromised RS. Of course, this does not prevent the client from contacting the RS before it can access the AS and ask about the tokens, but there is not much that can be done about it until the client is able to communicate with the AS.

3.1.2. Specific Recommendations

The following recommendations are useful to consider when implementing client introspection:

1- The AS should have a way to limit which clients are allowed to send introspection requests. This ensures that only clients that really do need the information are allowed access to it.

2- The "kid" header parameter as defined in [RFC7519] and [RFC8152] should be used when the token is encrypted in a structured information object such as a JSON Web Token (JWT) [RFC7519] or CBOR Web Token (CWT) [RFC8392]. The AS can store a key ID in this header that can be associated with the RS key used during encrypted token creation. If the AS does this when generating every encrypted token,

then it should always be able to decrypt that token on an introspection request coming from a client or from a RS.

This is needed because an encrypted token can only be decrypted if the proper key is known. When an RS performs introspection, the AS can use the identity of the RS as a hint to find the related key. However, if a client is performing the introspection request, the AS receiving the request needs more information to know what audience the encrypted token was issued for in order to decrypt it properly.

3- Only the value of the "active" parameter should be returned for introspection requests coming from clients. An introspection response has several parameters, but all of them are optional except for the "active" parameter. The "active" parameter can be used to indicate that a token has been revoked, and does not provide any information about the claims, which the client should usually not need. This prevents the disclosure of additional information to the client.

3.1.3. Alternatives

An alternative way to handle token revocation would be to prevent the AS from issuing more tokens for the same RS/audience, and for the client to request a new token each time it is in range of the AS. In this case, tokens would not be revoked, but rather clients would be implicitly notified to no longer contact a specific RS. However, this has at least two downsides. First, a client would have to request a new token each time it is in range of an AS, constantly, to be able to detect token revocation by getting the token request to be denied. This could lead to many tokens issued to the same client and for the same RS in a short period of time, which may not be even used in that timeframe. In addition, this generates additional traffic in an already constrained network. Second, client would be interpreting a denial to issue a token from an AS as a warning not to contact that RS anymore. This could lead the client to dump a previous token that it has for that RS, to prevent potentially dangerous contact with it. However, the denial may be for other reasons, but there is no way to differentiate when denying a token request to a client. Thus, a client may end up dumping working tokens because of a potentially different issue with new token generation. In summary, this option depends on the client making too many assumptions to successfully prevent it from accessing a compromised RS. Using client introspection to detect revoked tokens is a much simpler and direct way of handling this issue.

Another similar alternative to revoking tokens is to issue tokens with very short lifetimes. In this case, even if a device having a token is compromised, the short lifetime will make that token expire

quickly, making revocation notifications unnecessary. The main problem with this option in disadvantaged networks is that clients will not often be in range of the AS that issues the tokens or of the RS they want to use the token with. Thus, if tokens have very short lifetimes, they may not last long enough for a client to actually send that token to the RS it needs to contact. Or even if it does, if the token expires shortly afterwards, the client will not be able to contact that or other RS in the same audience again until it comes in range of the AS to obtain a new token. Thus, in this type of environments, the lifetime of a token must be carefully balanced in relation to its intended use and the frequency the devices will be in range of each other.

4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

5. Security Considerations

There are some potential security issues with the recommendations described in this document. Because the AS would accept introspection requests from a client, claim information associated to the tokens and not intended for a client could be sent back to it in a response. The recommendations above explicitly indicate to only send the "active" parameter as the response to this type of request, but it is still up to the implementation to do this properly, and to properly identify a device as a client (or more specifically as a device to send limited information to in a reply). If this is properly done, compromised or rogue clients sending introspection requests would not be able to obtain more information than the token active status from these types of introspection requests.

6. Acknowledgements

7. Normative References

- [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>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.

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

Authors' Addresses

Sebastian Echeverria
CMU SEI

Ludwig Seitz
RISE

Dan Klinedinst
CMU SEI

Grace Lewis
CMU SEI

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

G. Selander
Ericsson AB
S. Raza
RISE
M. Furuhed
Nexus
M. Vucinic
INRIA
T. Claeys
May 05, 2021

Protecting EST Payloads with OSCORE
draft-selander-ace-coap-est-oscore-05

Abstract

This document specifies public-key certificate enrollment procedures protected with lightweight application-layer security protocols suitable for Internet of Things (IoT) deployments. The protocols leverage payload formats defined in Enrollment over Secure Transport (EST) and existing IoT standards including the Constrained Application Protocol (CoAP), Concise Binary Object Representation (CBOR) and the CBOR Object Signing and Encryption (COSE) format.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 6, 2021.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Operational Differences with EST-coaps	4
2. Terminology	5
3. Authentication	5
3.1. EDHOC	5
3.2. Certificate-based Authentication	6
3.3. Channel Binding	6
3.4. Optimizations	6
3.5. RPK-based Trust Anchors	7
4. Protocol Design and Layering	7
4.1. Discovery and URI	8
4.2. Distribution of RPKs	8
4.3. Mandatory/optional EST Functions	8
4.4. Payload formats	9
4.5. Message Bindings	10
4.6. CoAP response codes	11
4.7. Message fragmentation	11
4.8. Delayed Responses	11
5. HTTP-CoAP Proxy	11
6. Security Considerations	12
7. Privacy Considerations	12
8. IANA Considerations	12
9. Acknowledgments	12
10. References	12
10.1. Normative References	12
10.2. Informative References	13
Appendix A. Other Authentication Methods	15
A.1. TTP Assisted Authentication	16
A.2. PSK Based Authentication	17
Appendix B. CBOR Encoding of EST Payloads	17
B.1. Distribution of CA Certificates (/crt)	18
B.2. Enrollment/Re-enrollment of Clients (/sen, /sren)	18
B.2.1. CBOR Certificate Request Examples	19
B.2.2. ASN.1 Certificate Request Examples	19
Authors' Addresses	21

1. Introduction

One of the challenges with deploying a Public Key Infrastructure (PKI) for the Internet of Things (IoT) is certificate enrollment, because existing enrollment protocols are not optimized for constrained environments [RFC7228].

One optimization of certificate enrollment targeting IoT deployments is specified in EST-coaps ([I-D.ietf-ace-coap-est]), which defines a version of Enrollment over Secure Transport [RFC7030] for transporting EST payloads over CoAP [RFC7252] and DTLS [RFC6347], instead of secured HTTP.

This document describes a method for protecting EST payloads over CoAP or HTTP with OSCORE [RFC8613]. OSCORE specifies an extension to CoAP which protects the application layer message and can be applied independently of how CoAP messages are transported. OSCORE can also be applied to CoAP-mappable HTTP which enables end-to-end security for mixed CoAP and HTTP transfer of application layer data. Hence EST payloads can be protected end-to-end independent of underlying transport and through proxies translating between between CoAP and HTTP.

OSCORE is designed for constrained environments, building on IoT standards such as CoAP, CBOR [RFC8949] and COSE [RFC8152], and has in particular gained traction in settings where message sizes and the number of exchanged messages needs to be kept at a minimum, such as 6TiSCH [I-D.ietf-6tisch-minimal-security], or for securing multicast CoAP messages [I-D.ietf-core-oscore-groupcomm]. Where OSCORE is implemented and used for communication security, the reuse of OSCORE for other purposes, such as enrollment, reduces the code footprint.

In order to protect certificate enrollment with OSCORE, the necessary keying material (notably, the OSCORE Master Secret, see [RFC8613]) needs to be established between EST-oscore client and EST-oscore server. For this purpose we assume the use of the lightweight authenticated key exchange protocol EDHOC [I-D.ietf-lake-edhoc]. Other methods for key establishment are described in Appendix A.

Other ways to optimize the performance of certificate enrollment and certificate based authentication described in this draft include the use of:

- o Compact representations of X.509 certificates (see [I-D.mattsson-cose-cbor-cert-compress])
- o Certificates by reference (see [I-D.ietf-cose-x509])

- o Compact representations of EST payloads (see Appendix B)

1.1. Operational Differences with EST-coaps

The protection of EST payloads defined in this document builds on EST-coaps [I-D.ietf-ace-coap-est] but transport layer security is replaced, or complemented, by protection of the transfer- and application layer data (i.e., CoAP message fields and payload). This specification deviates from EST-coaps in the following respects:

- o The DTLS record layer is replaced, or complemented, with OSCORE.
- o The DTLS handshake is replaced, or complemented, with the lightweight authenticated key exchange protocol EDHOC [I-D.ietf-lake-edhoc], and makes use of the following features:
 - * Authentication based on certificates is complemented with authentication based on raw public keys.
 - * Authentication based on signature keys is complemented with authentication based on static Diffie-Hellman keys, for certificates/raw public keys.
 - * Authentication based on certificate by value is complemented with authentication based on certificate/raw public keys by reference.
- o One new EST function, /rpks, is defined for installation of compact explicit TAs in the EST client.
- o The EST payloads protected by OSCORE can be proxied between constrained networks supporting CoAP/CoAPs and non-constrained networks supporting HTTP/HTTPs with a CoAP-HTTP proxy protection without any security processing in the proxy (see Section 5). The concept "Registrar" and its required trust relation with EST server as described in Section 6 of [I-D.ietf-ace-coap-est] is therefore redundant.

So, while the same authentication scheme (Diffie-Hellman key exchange authenticated with transported certificates) and the same EST payloads as EST-coaps also apply to EST-oscore, the latter specifies other authentication schemes and a new matching EST function. The reason for these deviations is that a significant overhead can be removed in terms of message sizes and round trips by using a different handshake, public key type or transported credential, and those are independent of the actual enrollment procedure.

Appendix A discusses yet other authentication and secure communication methods.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

This document uses terminology from [I-D.ietf-ace-coap-est] which in turn is based on [RFC7030] and, in turn, on [RFC5272].

The term "Trust Anchor" follows the terminology of [RFC6024]: "A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative." One example of specifying more compact alternatives to X.509 certificates for exchanging trust anchor information is provided by the TrustAnchorInfo structure of [RFC5914], the mandatory parts of which essentially is the SubjectPublicKeyInfo structure [RFC5280], i.e., an algorithm identifier followed by a public key.

3. Authentication

This specification replaces the DTLS handshake in EST-coaps with the lightweight authenticated key exchange protocol EDHOC [I-D.ietf-lake-edhoc]. During initial enrollment the EST-oscore client and server run EDHOC [I-D.ietf-lake-edhoc] to authenticate and establish the OSCORE security context with which the EST payloads are protected.

EST-oscore clients and servers MUST perform mutual authentication. The EST server and EST client are responsible for ensuring that an acceptable cipher suite is negotiated. The client MUST authenticate the server before accepting any server response. The server MUST authenticate the client and provide relevant information to the CA for decision about issuing a certificate.

3.1. EDHOC

EDHOC supports authentication with certificates/raw public keys (referred to as "credentials"), and the credentials may either be transported in the protocol, or referenced. This is determined by the identifier of the credential of the endpoint, ID_CRED_x for x= Initiator/Responder, which is transported in an EDHOC message. This

identifier may be the credential itself (in which case the credential is transported), or a pointer such as a URI to the credential (e.g., x5t, see [I-D.ietf-cose-x509]) or some other identifier which enables the receiving endpoint to retrieve the credential.

3.2. Certificate-based Authentication

EST-oscore, like EST-coaps, supports certificate-based authentication between EST client and server. In this case the client **MUST** be configured with an Implicit or Explicit Trust Anchor (TA) [RFC7030] database, enabling the client to authenticate the server. During the initial enrollment the client **SHOULD** populate its Explicit TA database and use it for subsequent authentications.

The EST client certificate **SHOULD** conform to [RFC7925]. The EST client and/or EST server certificate **MAY** be a (natively signed) CBOR certificate [I-D.mattsson-cose-cbor-cert-compress].

3.3. Channel Binding

The [RFC5272] specification describes proof-of-possession as the ability of a client to prove its possession of a private key which is linked to a certified public key. In case of signature key, a proof-of-possession is generated by the client when it signs the PKCS#10 Request during the enrollment phase. Connection-based proof-of-possession is **OPTIONAL** for EST-oscore clients and servers.

When desired the client can use the EDHOC-Exporter API to extract channel-binding information and provide a connection-based proof-of-possession. Channel-binding information is obtained as follows

```
edhoc-unique = EDHOC-Exporter("EDHOC Unique", length),
```

where length equals the desired length of the edhoc-unique byte string. The client then adds the edhoc-unique byte string as a challengePassword (see Section 5.4.1 of [RFC2985]) in the attributes section of the PKCS#10 Request to prove to the server that the authenticated EDHOC client is in possession of the private key associated with the certification request, and signed the certification request after the EDHOC session was established.

3.4. Optimizations

- o The last message of the EDHOC protocol, message_3, **MAY** be combined with an OSCORE request, enabling authenticated Diffie-Hellman key exchange and a protected CoAP request/response (which may contain an enrolment request and response) in two round trips [I-D.palombini-core-oscore-edhoc].

- o The certificates MAY be compressed, e.g. using the CBOR encoding defined in [I-D.mattsson-cose-cbor-cert-compress].
- o The certificate MAY be referenced instead of transported [I-D.ietf-cose-x509]. The EST-oscore server MAY use information in the credential identifier field of the EDHOC message (ID_CRED_x) to access the EST-oscore client certificate, e.g., in a directory or database provided by the issuer. In this case the certificate may not need to be transported over a constrained link between EST client and server.
- o Conversely, the response to the PKCS#10 request MAY be a reference to the enrolled certificate rather than the certificate itself. The EST-oscore server MAY in the enrolment response to the EST-oscore client include a pointer to a directory or database where the certificate can be retrieved.

3.5. RPK-based Trust Anchors

A trust anchor is commonly a self-signed certificate of the CA public key. In order to reduce transport overhead, the trust anchor could be just the CA public key and associated data (see Section 2), e.g., the SubjectPublicKeyInfo, or a public key certificate without the signature. In either case they can be compactly encoded, e.g. using CBOR encoding [I-D.mattsson-cose-cbor-cert-compress]. A client MAY request an unsigned trust anchors using the /rpks function (see Section 4.2).

Client authentication can be performed with long-lived RPKs installed by the manufacturer. Re-enrollment requests can be authenticated through a valid certificate issued previously by the EST-oscore server or by using the key material available in the Implicit TA database.

TODO: Sanity check this. Review the use of Implicit TA vs. Explicit TA.

4. Protocol Design and Layering

EST-oscore uses CoAP [RFC7252] and Block-Wise [RFC7959] to transfer EST messages in the same way as [I-D.ietf-ace-coap-est]. Instead of DTLS record layer, OSCORE [RFC8613] is used to protect the EST payloads. Figure 1 below shows the layered EST-oscore architecture.

EST request/response messages			
CoAP with OSCORE		HTTP with OSCORE	
UDP	DTLS/UDP	TCP	TLS/TCP

Figure 1: EST protected with OSCORE.

EST-oscore follows much of the EST-coaps and EST design.

4.1. Discovery and URI

The discovery of EST resources and the definition of the short EST-coaps URI paths specified in Section 5.1 of [I-D.ietf-ace-coap-est], as well as the new Resource Type defined in Section 9.1 of [I-D.ietf-ace-coap-est] apply to EST-oscore. Support for OSCORE is indicated by the "osc" attribute defined in Section 9 of [RFC8613], for example:

```
REQ: GET /.well-known/core?rt=ace.est.sen

RES: 2.05 Content
</est>; rt="ace.est";osc
```

4.2. Distribution of RPKs

The EST client can request a copy of the current CA public keys.

TODO: Map relevant parts of section 4.1 of RFC 7030 and other EST function related content from RFC7030 and EST-coaps.

RATIONALE: EST-coaps provides the /crts operation. A successful request from the client to this resource will be answered with a bag of certificates which is subsequently installed in the Explicit TA. Motivated by the specification of more compact trust anchors (see Section 2) we define here the new EST function /rpks which returns a set of RPKs to be installed in the Explicit TA database.

4.3. Mandatory/optional EST Functions

The EST-oscore specification has the same set of required-to-implement functions as EST-coaps. The content of Table 1 is adapted from Section 5.2 in [I-D.ietf-ace-coap-est] and uses the updated URI paths (see Section 4.1).

EST functions	EST-oscore implementation
/crtS	MUST
/sen	MUST
/sren	MUST
/skg	OPTIONAL
/skc	OPTIONAL
/att	OPTIONAL

Table 1: Mandatory and optional EST-oscore functions

TODO: Add /rpks OPTIONAL

4.4. Payload formats

Similar to EST-coaps, EST-oscore allows transport of the ASN.1 structure of a given Media-Type in binary format. In addition, EST-oscore uses the same CoAP Content-Format Options to transport EST requests and responses. Table 2 summarizes the information from Section 5.3 in [I-D.ietf-ace-coap-est].

URI	Content-Format	#IANA
/crtS	N/A (req)	-
	application/pkix-cert (res)	287
	application/pkcs-7-mime;smime-type=certs-only (res)	281
/sen	application/pkcs10 (req)	286
	application/pkix-cert (res)	287
	application/pkcs-7-mime;smime-type=certs-only (res)	281

/sren	application/pkcs10 (req)	286
	application/pkix-cert (res)	287
	application/pkcs-7-mime;smime-type=certs-only (res)	281
/skg	application/pkcs10 (req)	286
	application/multipart-core (res)	62
/skc	application/pkcs10 (req)	286
	application/multipart-core (res)	62
/att	N/A (req)	-
	application/csrattrs (res)	285

Table 2: EST functions and there associated Media-Type and IANA numbers

NOTE: CBOR is becoming a de facto encoding scheme in IoT settings. There is already work in progress on CBOR encoding of X.509 certificates [I-D.mattsson-cose-cbor-cert-compress], and this can be extended to other EST messages, see Appendix B.

4.5. Message Bindings

The EST-oscore message characteristics are identical to those specified in Section 5.4 of [I-D.ietf-ace-coap-est]. It is RECOMMENDED that

- o The EST-oscore endpoints support delayed responses
- o The endpoints supports the following CoAP options: OSCORE, Uri-Host, Uri-Path, Uri-Port, Content-Format, Block1, Block2, and Accept.

- o The EST URLs based on https:// are translated to coap://, but with mandatory use of the CoAP OSCORE option.

4.6. CoAP response codes

See Section 5.5 in [I-D.ietf-ace-coap-est].

4.7. Message fragmentation

The EDHOC key exchange is optimized for message overhead, in particular the use of static DH keys instead of signature keys for authentication (e.g., method 3 of [I-D.ietf-lake-edhoc]). Together with various measures listed in this document such as CBOR payloads (Appendix B), CBOR certificates [I-D.mattsson-cose-cbor-cert-compress], certificates by reference (Section 3.4), and trust anchors without signature (Section 3.5), a significant reduction of message sizes can be achieved.

Nevertheless, depending on application, the protocol messages may become larger than available frame size resulting in fragmentation and, in resource constrained networks such as IEEE 802.15.4 where throughput is limited, fragment loss can trigger costly retransmissions.

It is RECOMMENDED to prevent IP fragmentation, since it involves an error-prone datagram reconstitution. To limit the size of the CoAP payload, this specification mandates the implementation of CoAP option Block1 and Block2 fragmentation mechanism [RFC7959] as described in Section 5.6 of [I-D.ietf-ace-coap-est].

4.8. Delayed Responses

See Section 5.7 in [I-D.ietf-ace-coap-est].

5. HTTP-CoAP Proxy

As noted in Section 6 of [I-D.ietf-ace-coap-est], in real-world deployments, the EST server will not always reside within the CoAP boundary. The EST-server can exist outside the constrained network in a non-constrained network that supports HTTP but not CoAP, thus requiring an intermediary CoAP-to-HTTP proxy.

Since OSCORE is applicable to CoAP-mappable HTTP (see Section 11 of [RFC8613]) the EST payloads can be protected end-to-end between EST client and EST server independent of transport protocol or potential transport layer security which may need to be terminated in the proxy, see Figure 2. Therefore the concept "Registrar" and its

required trust relation with EST server as described in Section 6 of [I-D.ietf-ace-coap-est] is redundant.

The mappings between CoAP and HTTP referred to in Section 9.1 of [I-D.ietf-ace-coap-est] apply, and additional mappings resulting from the use of OSCORE are specified in Section 11 of [RFC8613].

OSCORE provides end-to-end security between EST Server and EST Client. The use of TLS and DTLS is optional.

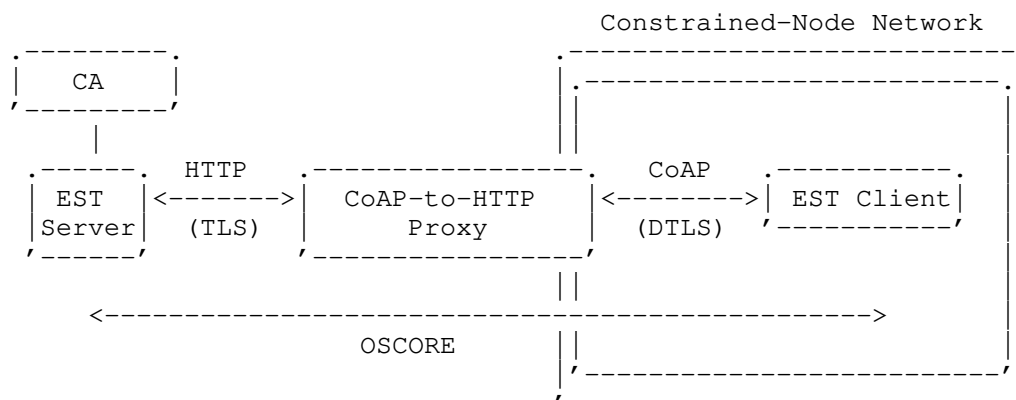


Figure 2: CoAP-to-HTTP proxy at the CoAP boundary.

6. Security Considerations

TBD

7. Privacy Considerations

TBD

8. IANA Considerations

9. Acknowledgments

10. References

10.1. Normative References

[I-D.ietf-ace-coap-est]

Stok, P. V. D., Kampanakis, P., Richardson, M. C., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-18 (work in progress), January 2020.

- [I-D.ietf-lake-edhoc]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-ietf-lake-edhoc-06 (work in progress), April 2021.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [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>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.
- [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>>.

10.2. Informative References

- [I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", draft-ietf-6tisch-minimal-security-15 (work in progress), December 2019.

- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-40 (work in progress), April 2021.
- [I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-18 (work in progress), April 2021.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-11 (work in progress), February 2021.
- [I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", draft-ietf-cose-x509-08 (work in progress), December 2020.
- [I-D.mattsson-cose-cbor-cert-compress]
Raza, S., Hoeglund, J., Selander, G., Mattsson, J. P., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", draft-mattsson-cose-cbor-cert-compress-08 (work in progress), February 2021.
- [I-D.palombini-core-oscore-edhoc]
Palombini, F., Tiloca, M., Hoeglund, R., Hristozov, S., and G. Selander, "Combining EDHOC and OSCORE", draft-palombini-core-oscore-edhoc-02 (work in progress), February 2021.
- [RFC2985] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes and Attribute Types Version 2.0", RFC 2985, DOI 10.17487/RFC2985, November 2000, <<https://www.rfc-editor.org/info/rfc2985>>.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<https://www.rfc-editor.org/info/rfc2986>>.

- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<https://www.rfc-editor.org/info/rfc5272>>.
- [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>>.
- [RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <<https://www.rfc-editor.org/info/rfc5914>>.
- [RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <<https://www.rfc-editor.org/info/rfc6024>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [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>>.

Appendix A. Other Authentication Methods

In order to protect certificate enrollment with OSCORE, the necessary keying material (notably, the OSCORE Master Secret, see [RFC8613]) needs to be established between EST-oscore client and EST-oscore server. In this appendix we analyse alternatives to EDHOC, which was assumed in the body of this specification.

A.1. TTP Assisted Authentication

Trusted third party (TTP) based provisioning, such as the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile] assumes existing security associations between the client and the TTP, and between the server and the TTP. This setup allows for reduced message overhead and round trips compared to the full-fledged EDHOC key exchange. Following the ACE terminology the TTP plays the role of the Authorization Server (AS), the EST-oscore client corresponds to the ACE client and the EST-oscore server is the ACE Resource Server (RS).

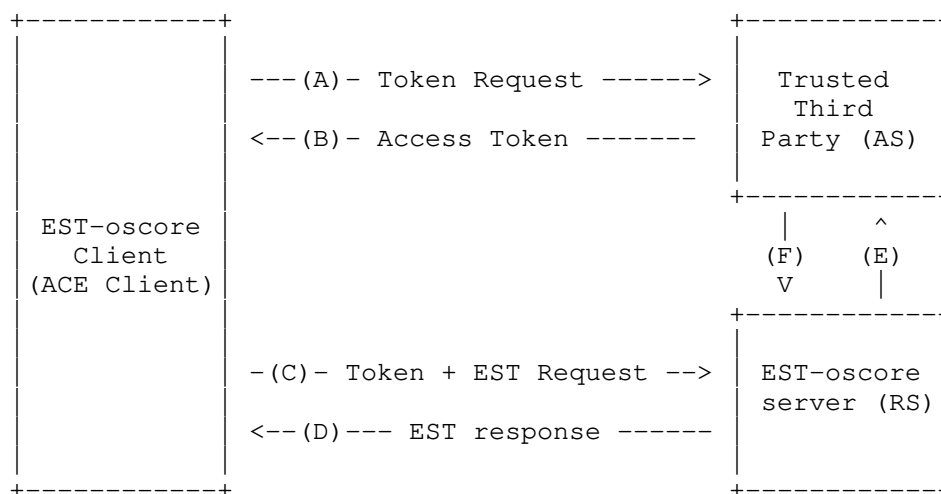


Figure 3: Accessing EST services using a TTP for authenticated key establishment and authorization.

During initial enrollment the EST-oscore client uses its existing security association with the TTP, which replaces the Implicit TA database, to establish an authenticated secure channel. The [I-D.ietf-ace-oscore-profile] ACE profile RECOMMENDS the use of OSCORE between client and TTP (AS), but TLS or DTLS MAY be used additionally or instead. The client requests an access token at the TTP corresponding the EST service it wants to access. If the client request was invalid, or not authorized according to the local EST policy, the AS returns an error response as described in Section 5.6.3 of [I-D.ietf-ace-oauth-authz]. In its responses the TTP (AS) SHOULD signal that the use of OSCORE is REQUIRED for a specific access token as indicated in section 4.3 of [I-D.ietf-ace-oscore-profile]. This means that the EST-oscore client MUST use OSCORE towards all EST-oscore servers (RS) for which this

access token is valid, and follow Section 4.3 in [I-D.ietf-ace-oscore-profile] to derive the security context to run OSCORE. The ACE OSCORE profile RECOMMENDS the use of CBOR web token (CWT) as specified in [RFC8392]. The TTP (AS) MUST also provision an OSCORE security context to the EST-oscore client and EST-oscore server (RS), which is then used to secure the subsequent messages between the client and the server. The details on how to transfer the OSCORE contexts are described in section 3.2 of [I-D.ietf-ace-oscore-profile].

Once the client has retrieved the access token it follows the steps in [I-D.ietf-ace-oscore-profile] to install the OSCORE security context and presents the token to the EST-oscore server. The EST-oscore server installs the corresponding OSCORE context and can either verify the validity of the token locally or request a token introspection at the TTP. In either case EST policy decisions, e.g., which client can request enrollment or reenrollment, can be implemented at the TTP. Finally the EST-oscore client receives a response from the EST-oscore server.

A.2. PSK Based Authentication

Another method to bootstrap EST services requires a pre-shared OSCORE security context between the EST-oscore client and EST-oscore server. Authentication using the Implicit TA is no longer required since the shared security context authenticates both parties. The EST-oscore client and EST-oscore server need access to the same OSCORE Master Secret as well as the OSCORE identifiers (Sender ID and Recipient ID) from which an OSCORE security context can be derived, see [RFC8613]. Some optional parameters may be provisioned if different from the default value:

- o an ID context distinguishing between different OSCORE security contexts to use,
- o an AEAD algorithm,
- o an HKDF algorithm,
- o a master salt, and
- o a replay window size.

Appendix B. CBOR Encoding of EST Payloads

Current EST based specifications transport messages using the ASN.1 data type declaration. It would be favorable to use a more compact representation better suitable for constrained device

implementations. In this appendix we list CBOR encodings of requests and responses of the mandatory EST functions (see Section 4.3).

B.1. Distribution of CA Certificates (/crts)

The EST client can request a copy of the current CA certificates. In EST-coaps and EST-oscore this is done using a GET request to /crts (with empty payload). The response contains a chain of certificates used to establish an Explicit Trust Anchor database for subsequent authentication of the EST server.

CBOR encoding of X.509 certificates is specified in [I-D.mattsson-cose-cbor-cert-compress]. CBOR encoding of certificate chains is specified below. This allows for certificates encoded using the CBOR certificate format, or as binary X.509 data wrapped as a CBOR byte string.

CDDL:

```
certificate chain = (  
    + certificate : bstr  
)  
certificate = x509_certificate / cbor_certificate
```

B.2. Enrollment/Re-enrollment of Clients (/sen, /sren)

Existing EST standards specify the enrollment request to be a PKCS#10 formatted message [RFC2986]. The essential information fields for the CA to verify are the following:

- o Information about the subject, here condensed to the subject common name,
- o subject public key, and
- o signature made by the subject private key.

CDDL:

```
certificate request = (  
    subject_common_name : bstr,  
    public_key : bstr  
    signature : bstr,  
    ? ( signature_alg : int, public_key_info : int )  
)
```

The response to the enrollment request is the subject certificate, for which CBOR encoding is specified in [I-D.mattsson-cose-cbor-cert-compress].

The same message content in request and response applies to re-enrollment.

TODO: PKCS#10 allows inclusion of attributes, which can be used to specify extension requests, see Section 5.4.2 of [RFC2985]. CBOR encoding of the challengePassword attribute needs to be defined (see Section 3.3). What other attributes are relevant?

B.2.1. CBOR Certificate Request Examples

Here is an example of CBOR encoding of certificate request as defined in the previous section.

114 bytes:

```
( h'0123456789ABCDEF0',  
  h'61eb80d2abf7d7e4139c86b87e42466f1b4220d3f7ff9d6a1ae298fb9adbb464',  
  h'30440220064348b9e52ee0da9f9884d8dd41248c49804ab923330e208a168172dca  
e1 27a02206a06c05957f1db8c4e207437b9ab7739cb857aa6dd9486627b8961606a2  
b68ae' )
```

In the example above the signature is generated on an ASN.1 data structure. To validate this, the receiver needs to reconstruct the original data structure. Alternatively, in native mode, the signature is generated on the profiled data structure, in which case the overall overhead is further reduced.

B.2.2. ASN.1 Certificate Request Examples

A corresponding certificate request of the previous section using ASN.1 is shown in Figure 4.

```

SEQUENCE {
  SEQUENCE {
    INTEGER 0
    SEQUENCE {
      SET {
        SEQUENCE {
          OBJECT IDENTIFIER commonName (2 5 4 3)
          UTF8String '01-23-45-67-89-AB-CD-F0'
        }
      }
    }
  }
  SEQUENCE {
    SEQUENCE {
      OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
      OBJECT IDENTIFIER prime256v1 (1 2 840 10045 3 1 7)
    }
    BIT STRING
      (65 byte public key)
  }
  SEQUENCE {
    OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
  }
  BIT STRING
    (64 byte signature)
}

```

Figure 4: ASN.1 Structure.

In Base64, 375 bytes:

```

-----BEGIN CERTIFICATE REQUEST-----
MIHcMIGEAgEAMCIxIDAeBgNVBAMMFzAxLTlZLTQ1LTg5LUFCLUNELUYwMFkw
EwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEYeuA0qv31+QTnIa4fkJGbxTCINP3/51q
GuKY+5rbtGSeZn3l8rVbU0jVEBwvKhAd98JeqgsuauGHRNWt2FqJ1aAAMAoGCCqG
SM49BAMCA0cAMEQCIAZDSLnlLuDan5iE2N1BJIxJgEq5IzMOIIoWgXLcrhJ6AiBq
BsBZV/HbjE4gdDe5q3c5y4V6pt2UhmJ7iWFgaitorg==
-----END CERTIFICATE REQUEST-----

```

In hex, 221 bytes:

```

3081dc30818402010030223120301e06035504030c1730312d32332d34352d36
372d38392d41422d43442d46303059301306072a8648ce3d020106082a8648ce
3d0301070342000461eb80d2abf7d7e4139c86b87e42466f1b4220d3f7ff9d6a
1ae298fb9adbb4649e667de5f2b55b5348d51015af2a101df7c25eaa0b2e6ae1
8744d5add85a89d5a000300a06082a8648ce3d04030203470030440220064348
b9e52ee0da9f9884d8dd41248c49804ab923330e208a168172dcae127a02206a
06c05957f1db8c4e207437b9ab7739cb857aa6dd9486627b8961606a2b68ae

```

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Shahid Raza
RISE

Email: shahid.raza@ri.se

Martin Furuhed
Nexus

Email: martin.furuhed@nexusgroup.com

Malisa Vucinic
INRIA

Email: malisa.vucinic@inria.fr

Timothy Claeys

Email: timothy.claeys@gmail.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 8, 2019

C. Sengul
Nominet
A. Kirby
Oxbotica
P. Fremantle
University of Portsmouth
April 6, 2019

MQTT-TLS profile of ACE
draft-sengul-ace-mqtt-tls-profile-04

Abstract

This document specifies a profile for the ACE (Authentication and Authorization for Constrained Environments) to enable authorization in an MQTT-based publish-subscribe messaging system. Proof-of-possession keys, bound to OAuth2.0 access tokens, are used to authenticate and authorize publisher and subscriber clients. The protocol relies on TLS for confidentiality and server authentication.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 8, 2019.

Copyright Notice

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

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	4
1.2. ACE-Related Terminology	4
1.3. MQTT-Related Terminology	4
2. Basic Protocol Interactions	5
2.1. Authorizing Connection Establishment	6
2.1.1. Client Authorization Server (CAS) and Authorization Server (AS) Interaction	7
2.1.2. Client Connection Request to the Broker	8
2.1.3. Token Validation	10
2.1.4. The Broker's Response to Client Connection Request	11
2.2. Authorizing PUBLISH Messages	12
2.2.1. PUBLISH Messages from the Publisher Client to the Broker	12
2.2.2. PUBLISH Messages from the Broker to the Subscriber Clients	12
2.3. Authorizing SUBSCRIBE Messages	13
2.4. Token Expiration	13
2.5. Handling Disconnections and Retained Messages	13
3. Improved Protocol Interactions with MQTT v5	14
3.1. Token Transport via Authentication Exchange (AUTH)	14
3.2. Authorization Errors and Client Re-authentication	16
4. IANA Considerations	17
5. Security Considerations	17
6. Privacy Considerations	18
7. References	18
7.1. Normative References	18
7.2. Informative References	19
Appendix A. Checklist for profile requirements	20
Appendix B. The Authorization Information Endpoint	21
Appendix C. Document Updates	21
Acknowledgements	22
Authors' Addresses	22

1. Introduction

This document specifies a profile for the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, clients and a resource server use MQTT to communicate. The protocol relies on TLS for communication security between entities. The basic protocol interactions follow MQTT v3.1.1 – the OASIS Standard

[MQTT-OASIS-Standard]. In addition, this document describes improvements to the basic protocol with the new MQTT v5.0 - the OASIS Standard [MQTT-OASIS-Standard-v5] (e.g., improved authentication exchange and error reporting). Both versions are expected to be supported in practice, and therefore, covered in this document.

MQTT is a publish-subscribe protocol and supports two main types of client operation: publish and subscribe. Once connected, a client can publish to multiple topics, and subscribe to multiple topics; however, for this document, these actions are described separately. The MQTT broker is responsible for distributing messages published by the publishers to the appropriate subscribers. Each publish message contains a topic, which is used by the broker to filter the subscribers for the message. Subscribers must subscribe to the topics to receive the corresponding messages.

In this document, message topics are treated as resources. Clients use an access token, bound to a key (the proof-of-possession key) to authorize with the MQTT broker their connection and publish/subscribe permissions to topics. In the context of this ACE profile, the MQTT broker acts as the resource server. To provide communication confidentiality and resource server authentication, TLS is used.

Clients use client authorization servers [I-D.ietf-ace-actors] to obtain tokens from the authorization server. The communication protocol between the client authorization server and the authorization server is assumed to be HTTPS. Also, if the broker supports token introspection, it is assumed to use HTTPS to communicate with the authorization server. These interfaces MAY be implemented using other protocols, e.g., CoAP or MQTT. This document makes the same assumptions as the Section 4 of the ACE framework [I-D.ietf-ace-oauth-authz] regarding client and RS registration with the AS and establishing of keying material.

This document describes the authorization of the following exchanges between publisher and subscriber clients, and the broker.

- o Connection establishment between the clients and the broker
- o Publish messages from the publishers to the broker, and from the broker to the subscribers
- o Subscribe messages from the subscribers to the broker

In Section 2, these exchanges are described based on the MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard]. These exchanges are also supported by the new MQTT v5 - the OASIS Standard

[MQTT-OASIS-Standard-v5]. Section 3 describes how they may be improved by the new MQTT v5.

1.1. Requirements Language

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.

1.2. ACE-Related Terminology

The terminology for entities in the architecture is defined in OAuth 2.0 RFC 6749 [RFC6749] and ACE actors [I-D.ietf-ace-actors], such as "Client" (C), "Resource Server" (RS) and "Authorization Server" (AS).

The term "endpoint" is used following its OAuth definition, to denote resources such as /token and /introspect at the AS.

The term "Resource" is used to refer to an MQTT "topic name," which is defined in Section 1.3. Hence, the "Resource Owner" is any entity that can authoritatively speak for the "topic".

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify" are taken from RFC 4949 [RFC4949].

1.3. MQTT-Related Terminology

The document describes message exchanges as MQTT protocol interactions. For additional information, please refer to the MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard] or the MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5].

Topic name

The label attached to an application message, which is matched to a subscription.

Topic filter

An expression that indicates interest in one or more topic names. Topic filters may include wildcards.

Subscription

A subscription comprises a Topic filter and a maximum quality of service (QoS).

Application Message

The data carried by the MQTT protocol. The data has an associated QoS level and a Topic name.

MQTT sends various control messages across a network connection. The following is not an exhaustive list and the control packets that are not relevant for authorization are not explained. These include, for instance, the PUBREL and PUBCOMP packets used in the 4-step handshake required for the QoS level 2.

CONNECT

Client request to connect to the broker. After a network connection is established, this is the first packet sent by a client.

CONNACK

The broker connection acknowledgment. The first packet sent from the broker to a client is a CONNACK packet. CONNACK packets contain return codes indicating either a success or an error state to a client.

PUBLISH

Publish packet that can be sent from a client to the broker, or from the broker to a client.

PUBACK

Response to PUBLISH packet with QoS level 1. PUBACK can be sent from the broker to a client or a client to the broker.

PUBREC

Response to PUBLISH packet with QoS level 2. PUBREC can be sent from the broker to a client or a client to the broker.

SUBSCRIBE

The client subscribe request.

SUBACK

Subscribe acknowledgment.

PINGREQ A ping request sent from a client to the broker. It signals to the broker that the client is alive, and is used to confirm that the broker is still alive.

2. Basic Protocol Interactions

This section describes the following exchanges between publisher and subscriber clients, the broker, and the authorization server according to the MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard]. These exchanges are compatible also with the

new MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5]. In addition, Section 3 describes how these exchanges may be improved with the MQTT v5.

- o Authorizing connection establishment between the clients and the broker
- o Authorizing publish messages from the publishers to the broker, and from the broker to the subscribers
- o Authorizing subscribe messages from the subscribers to the broker

Message topics are treated as resources. The publisher and subscriber clients are assumed to have identified the topics of interest out-of-band (topic discovery is not a feature of the MQTT protocol).

A connection request carries a token specifying the permissions that the client has (e.g., publish permission to a given topic). A resource owner can pre-configure policies at the AS that give clients publish or subscribe permissions to different topics.

2.1. Authorizing Connection Establishment

This section specifies how publishers and subscribers establish an authorized connection to an MQTT broker. The token request and response use the /token endpoint of the authorization server, as specified in Section 5 of the ACE framework [I-D.ietf-ace-oauth-authz].

Figure 1 shows the basic protocol flow during connection establishment. The step (C), client onboarding, is out of the scope of this document. Steps (E) and (F) are optional.

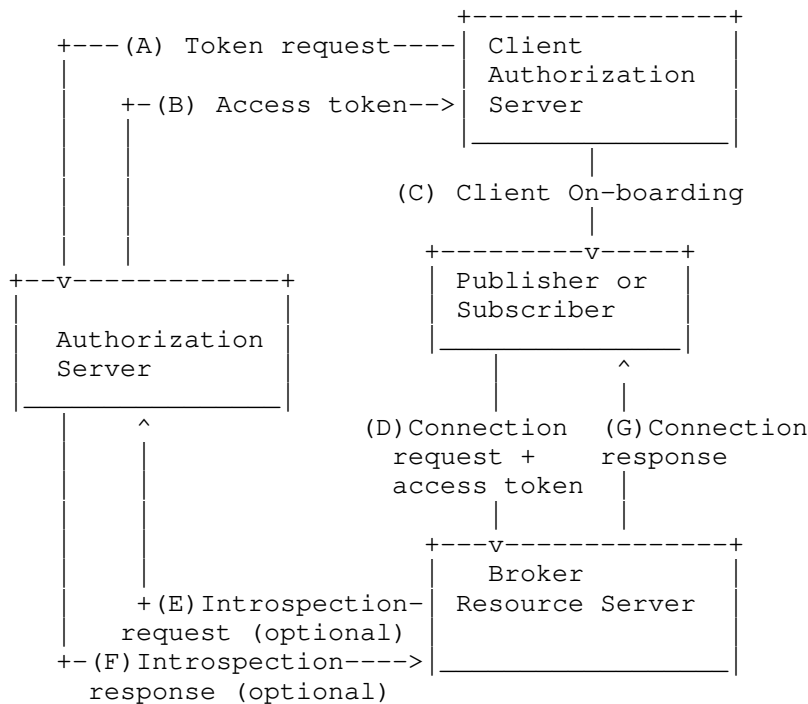


Figure 1: Connection establishment

2.1.1. Client Authorization Server (CAS) and Authorization Server (AS) Interaction

The first step in the protocol flow (Figure 1 (A)) is the token acquisition by the client authorization server (CAS) from the AS. If a client has enough resources and can support HTTPS, or optionally the AS supports MQTTS, these steps can instead be carried out by a client directly.

When requesting an access token from the AS, the CAS MAY include parameters in its request as defined in Section 5.6.1 of the ACE framework [I-D.ietf-ace-oauth-authz]. The content type is set to "application/json". The profile parameter is set to 'mqtt-tls'.

If the AS successfully verifies the access token request and authorizes the client for the indicated audience (e.g., RS) and scopes (e.g., publish/subscribe permissions over topics), the AS issues an access token (Figure 1 (B)). The response includes the parameters described in Section 5.6.2 of the ACE framework [I-D.ietf-ace-oauth-authz]. The included token is assumed to be Proof-of-Possession (PoP) token by default. Hence, a 'cnf' parameter

with a symmetric or asymmetric PoP key is returned. The token may be a reference, or a CBOR or JWT web token. Note that the 'cnf' parameter in the web tokens are to be consumed by the resource server and not the client. For more information on Proof of Possession semantics in JWTs see RFC 7800 [RFC7800] and for CWTs, see Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs) [I-D.ietf-ace-cwt-proof-of-possession].

In the case of an error, the AS returns error responses for HTTP-based interactions as ASCII codes in JSON content, as defined in Section 5.2 of RFC 6749 [RFC6749].

2.1.2. Client Connection Request to the Broker

Once the client acquires the token, it can use it to request an MQTT connection to the broker over a TLS session with server authentication (Figure 1 (D)). This section describes the client transporting the token to the broker (RS) via the CONNECT control message after the TLS handshake. This is similar to an earlier proposal by Fremantle et al. [fremantle14]. An improvement to this is presented in Section 3 for the MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5]. Alternatively, the token may be used for the TLS session establishment as described in the DTLS profile for ACE [I-D.gerdes-ace-dtls-authorize]. In this case, both the TLS PSK and RPK handshakes MAY be supported. This may additionally require that the client transports the token to the broker before the connection establishment. To this end, the broker MAY support /authz-info endpoint via the "authz-info" topic. Then, to transport the token, clients publish to "authz-info" topic unauthorized. The topic "authz-info" MUST be publish-only for clients (i.e., the clients are not allowed to subscribe to it). This option is described in more detail in Appendix B.

When the client wishes to connect to the broker, it uses the CONNECT message of MQTT. Figure 2 shows the structure of the MQTT CONNECT control message.

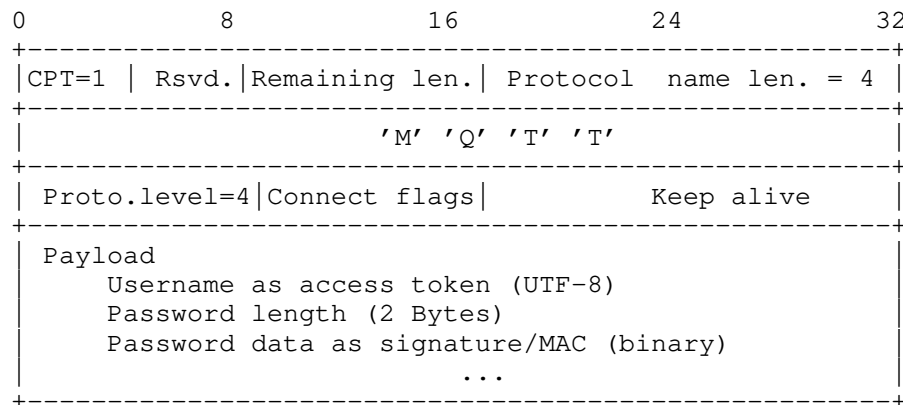


Figure 2: MQTT CONNECT control message. (CPT=Control Packet Type, Rsvd=Reserved, len.=length, Proto.=Protocol)

To communicate the necessary connection parameters, the Client uses the appropriate flags of the CONNECT message. Figure 3 shows how the MQTT connect flags MUST be set to initiate a connection with the broker.

User name flag	Pass. flag	Will retain	Will QoS	Will Flag	Clean	Rsvd.
1	1	X	X X	X	1	0

Figure 3: MQTT CONNECT flags. (Rsvd=Reserved)

To ensure that the client and the broker discard any previous session and start a new session, the Clean Session Flag MUST be set to 1.

The Will flag indicates that a Will message needs to be sent when a client disconnection occurs. The situations in which the Will message is published include disconnections due to I/O or network failures, and the server closing the networking connection due to a protocol error. The client may set the Will flag as desired (marked as 'X' in Figure 3). If the Will flag is set to 1 and the broker accepts the connection request, the broker must store the Will message, and publish it when the network connection is closed according to Will QoS and Will retain parameters, and MQTT Will management rules. Section 2.5 explains how the broker deals with the retained messages in further detail.

Finally, Username and Password flags MUST be set to 1 to ensure that the Payload of the CONNECT message includes both Username and Password fields.

The CONNECT message defaults to ACE for authentication and authorization. For the basic operation described in this section, the Username field MUST be set to the access token. The Password field MUST be set to the keyed message digest (MAC) or signature associated with the access token for proof-of-possession. The client MAY apply the PoP key either to the entire request by computing a keyed message digest (for symmetric key) or a digital signature (for asymmetric key). The CONNECT message is assumed to have enough randomness in the payload, and inside a TLS session (excluding the 0-RTT case) will not be exposed to a replay attack. When either cannot be guaranteed, the Password MAY also contain a nonce.

Section 3.1.3 of MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard] defines the MQTT Username as a UTF-8 encoded string, which is prefixed by a 2-byte length field followed by UTF-8 encoded character data up to 65535 bytes. Therefore an access token that is not a valid UTF-8 MUST be Base64 [RFC4648] encoded. (The MQTT Password allows binary data up to 65535 bytes, and so, does not require encoding.)

2.1.3. Token Validation

RS MUST verify the validity of the token. This validation MAY be done locally (e.g., in the case of a self-contained token) or the RS MAY send an introspection request to the AS. If introspection is used, this section follows similar steps to those described in Sections 5.7 of the ACE framework [I-D.ietf-ace-oauth-authz]. The communication between AS and RS MAY be HTTPS, but it, in every case, MUST be confidential, mutually authenticated and integrity protected.

The broker MUST check if the token is active either using 'exp' claim of the token or 'active' parameter of the introspection response.

The access token is constructed by the AS such that RS can associate the access token with the client key. This document assumes that the Access Token is a PoP token as described in [I-D.ietf-ace-oauth-authz]. Therefore, the necessary information is contained in the 'cnf' claim of the access token and may use either public or shared key approaches. The client uses the signature or the MAC in the password field to prove the possession of the key. The resource server validates the signature or the MAC over the contents of the packet, authenticating the client.

The broker uses the scope field in the token (or in the introspection result) to determine the publish and subscribe permissions for the client. If the Will flag is set, then the broker MUST check that the token allows the publication of the Will message too.

If the token is not self-contained and the broker uses token introspection, it MAY cache the validation result to decide whether to accept subsequent PUBLISH and SUBSCRIBE messages as these messages, which are sent after a connection set-up, do not contain access tokens. If the introspection result is not cached, then the RS needs to introspect the saved token for each request.

Scope strings SHOULD be encoded as a permission, followed by an underscore, followed by a topic filter. Two permissions apply to topics: 'publish' and 'subscribe'. An example scope field may contain multiple such strings, space delimited, e.g., 'publish_topic1 subscribe_topic2/#'. Hence, this access token would give 'publish' permission to the 'topic1', 'subscribe' permission to all the subtopics of 'topic2'.

Also, if present in the access token, RS must check that the 'iss' corresponds to AS, the 'aud' field (if not used to define topics) corresponds to RS. It also has to check whether 'nbf' and 'iat' claims are present and valid.

2.1.4. The Broker's Response to Client Connection Request

Based on the validation result (obtained either via local inspection or using the /introspection interface of the AS), the broker MUST send a CONNACK message to the client.

The broker responses may follow either the MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard] or the MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5], depending on which version(s) the broker supports.

In MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard], it is not possible to support AS discovery via sending a tokenless CONNECT message to the broker. This is because a CONNACK packet does not include a means to provide additional information to the client. Therefore, AS discovery needs to take place out-of-band. This is remedied in the MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5] and a solution is described in Section 3.

If the RS accepts the connection, it MUST store the token.

2.2. Authorizing PUBLISH Messages

2.2.1. PUBLISH Messages from the Publisher Client to the Broker

On receiving the PUBLISH message, the broker MUST use the type of message (i.e., PUBLISH) and the topic name in the message header to compare against the cached token or its introspection result.

If the client is allowed to publish to the topic, the RS must publish the message to all valid subscribers of the topic. The broker may also return an acknowledgment message if the QoS level is greater than or equal to 1.

In case of a failure, it is not possible to return an error in MQTT v3.1.1 – the OASIS Standard [MQTT-OASIS-Standard]. Acknowledgement messages only indicate success. In the case of an authorization error, the broker SHOULD disconnect the client. Otherwise, it MUST ignore the PUBLISH message. Also, DISCONNECT messages are only sent from a client to the broker. So, server disconnection needs to take place below the application layer. However, in MQTT v5 – the OASIS Standard [MQTT-OASIS-Standard-v5], it is possible to indicate failure and provide a reason code. Section 3 describes in more detail how MQTT v5 handles PUBLISH authorization errors.

2.2.2. PUBLISH Messages from the Broker to the Subscriber Clients

To forward PUBLISH messages to the subscribing clients, the broker identifies all the subscribers that have valid matching topic subscriptions (i.e., the tokens are valid, and token scopes allow a subscription to the particular topic name). The broker sends a PUBLISH message with the topic name and the topic message to all the valid subscribers.

In MQTT, after connection establishment, there is no way to inform a client that an authorization error has occurred for previously subscribed topics, e.g., token expiry. In the case of an authorization error, the broker disconnects the client. In the MQTT v3.1.1 – the OASIS Standard [MQTT-OASIS-Standard], the MQTT DISCONNECT messages are only sent from a client to the broker. Therefore, the server disconnection needs to take place below the application layer. In MQTT v5 – the OASIS Standard [MQTT-OASIS-Standard-v5], a server-side DISCONNECT message is possible and described in Section 3.

2.3. Authorizing SUBSCRIBE Messages

In MQTT, a SUBSCRIBE message is sent from a client to the broker to create one or more subscriptions to one or more topics. The SUBSCRIBE message may contain multiple topic filters. The topic filters may include wildcard characters.

On receiving the SUBSCRIBE message, the broker MUST use the type of message (i.e., SUBSCRIBE) and the topic filter in the message header to compare against the stored token or introspection result.

As a response to the SUBSCRIBE message, the broker issues a SUBACK message. For each topic filter, the SUBACK packet includes a return code matching the QoS level for the corresponding topic filter. In the case of failure, the return code, in MQTT v3.1.1, must be 0x80 indicating 'Failure'. In MQTT v5, the appropriate return code is 0x87, indicating that the client is 'Not authorized'. Note that, in both MQTT versions, a reason code is returned for each topic filter. Therefore, the client may receive success codes for a subset of its topic filters, while being unauthorized for the rest.

2.4. Token Expiration

The broker MUST check for token expiration whenever a CONNECT, PUBLISH or SUBSCRIBE message is received or sent. The broker SHOULD check for token expiration on receiving a PINGREQUEST message. This may allow for early detection of a token expiry.

The token expiration is checked by checking the 'exp' claim of a CWT/JWT or via performing an introspection request with the Authorization server as described in Section 5.7 of the ACE framework [I-D.ietf-ace-oauth-authz]. In the basic operation, token expirations MAY lead to disconnecting the associated client. However, in MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5], better error handling and re-authentication are possible. This is explained in more detail in Section 3.

2.5. Handling Disconnections and Retained Messages

According to MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard], only Client DISCONNECT messages are allowed. In MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5], server-side DISCONNECT messages are possible, allowing to return '0x87 Not Authorized' return code to the client.

In the case of a DISCONNECT, due to the Clean Session flag, the broker deletes all session state but MUST keep the retained messages. By setting a RETAIN flag in a PUBLISH message, the publisher

indicates to the broker that it should store the most recent message for the associated topic. Hence, the new subscribers can receive the last sent message from the publisher for that particular topic without waiting for the next PUBLISH message. In the case of a disconnection, the broker MUST continue publishing the retained messages as long as the associated tokens are valid.

In case of disconnections due to network errors or server disconnection due to a protocol error (which includes authorization errors), the Will message must be sent if the client supplied a Will in the CONNECT request message. The token provided in the CONNECT request must cover the Will topic. The Will message MUST be published to the Will topic when the network connection is closed regardless of whether the corresponding token has expired.

3. Improved Protocol Interactions with MQTT v5

In the new MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5], several new capabilities are introduced, which enable better integration with ACE. The newly enhanced authentication and re-authentication methods support a wider range of authentication flows beyond username and password. With the MQTT v5, there is a clearly defined approach for using token-based authorization. Also, it is possible for a client to request a re-authentication avoiding disconnection. Finally, MQTT v5 generally improves error reporting, enabling better response to authorization failures during publishing messages to the subscribers.

3.1. Token Transport via Authentication Exchange (AUTH)

To initiate the authentication and authorization flow, as before, the CAS initiates the token request as in Section 2.1. When the client wishes to connect to the RS (broker), it uses the CONNECT message of MQTT. Figure 4 shows the structure of the MQTT CONNECT control message used in MQTT v5.

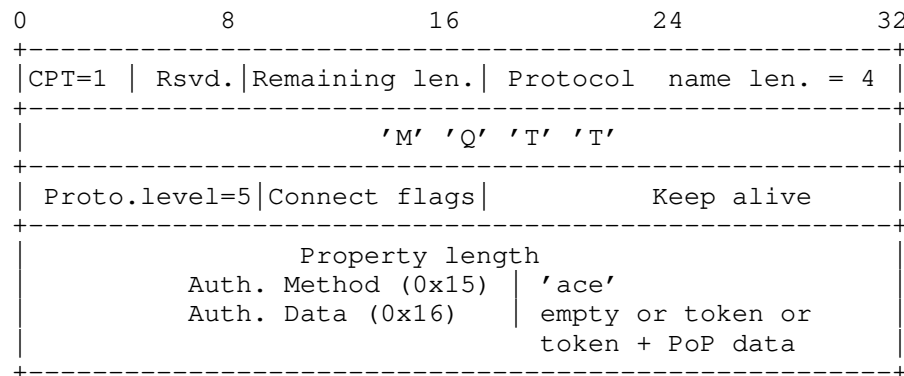


Figure 4: MQTT CONNECT control message. (CPT=Control Packet Type, Rsvd=Reserved, len.=length, Proto.=Protocol)

To communicate the necessary connection parameters, the client uses the appropriate flags of the CONNECT message. To achieve a clean session (i.e., the session should start without an existing session), the new MQTT v5 session flags MUST be set appropriately: the Clean Start Flag MUST be set to 1 and Session Expiry Interval MUST be set to 0.

With the enhanced authentication capabilities, it is not necessary to overload the username and password fields in the CONNECT message for ACE authentication. Nevertheless, the RS MUST support both methods for supporting the token: (1) Token transport via username and password and (2) using the new AUTH (Authentication Exchange) method. The token transport via username and password is as described in Section 2.1.2. The rest of this section describes the AUTH method.

To use the AUTH method, the username flag MUST be set to 0, and the password flag MUST be set to 0. The client can set the Authentication Method as a property of a CONNECT packet by setting Auth Properties (with the property identifier 0x15). The client must MUST set the UTF-8 encoded string containing the name of the authentication method as 'ace'. If the RS does not support this profile, it sends a CONNACK with a Reason Code of '0x8C (Bad authentication method)'

The Authentication Method is followed by the Authentication Data, which has a property identifier 0x16. Authentication data is binary data and is defined by the authentication method. The RS MAY support different implementations for transporting the authentication data. The first option is that Authentication data contains both the token and the keyed message digest (MAC) or signature as described in Section 2.1.2. The encoding of this field MAY use CBOR and COSE. In

this case, the token validation proceeds as described in Section 2.1.3 and the server responds with a CONNACK. The reason code of the CONNACK is '0x00 (Success)' if the authentication is successful. In case of an invalid PoP token, the CONNACK reason code is '0x87 (Not Authorized)'.

The second option that RS may accept is a challenge/response protocol. If the Authentication Data only includes the token, the RS MUST respond with an AUTH packet, with the Authenticate Reason Code set to '0x18 (Continue Authentication)'. This packet includes the Authentication Method, which MUST be set to 'ace' and Authentication Data. The Authentication Data MUST NOT be empty and contains a challenge for the client. The client responds to this with an AUTH packet, with a reason code '0x18 (Continue Authentication)'. Similarly, the client packet sets the Authentication Method to 'ace'. The Authentication Data in the client's response contains the signature or MAC computed over the RS's challenge. To this, the server responds with a CONNACK and return code '0x00 (Success)' if the authentication is successful. In case of an invalid PoP token, the CONNACK reason code is '0x87 (Not Authorized)'.

Finally, this document allows the CONNECT message to have an empty Authentication Data field. This is the AS discovery option and the RS responds with the CONNACK reason code '0x87 (Not Authorized)' and includes a User Property for the AS information. AS Information contains the absolute URI of AS, and MAY also contain a cnonce as described in the Section 5.1 of the ACE framework [I-D.ietf-ace-oauth-authz]. This information MAY be CBOR encoded.

3.2. Authorization Errors and Client Re-authentication

MQTT v5 allows better error reporting. To take advantage of this for PUBLISH messages, the QoS level should be set to greater than or equal to 1. This guarantees that RS responds with either a PUBACK or PUBREC packet with reason code '0x87 (Not authorized)' in the case of an authorization error. Similarly, for the SUBSCRIBE case, the SUBACK packet has a reason code set to '0x87 (Not authorized)' for the unauthorized topic(s). When RS is forwarding PUBLISH messages to the subscribed clients, it may discover that some of the subscribers are no more authorized due to expired tokens. In this case, the RS SHOULD send a DISCONNECT message with the reason code '0x87 (Not authorized)'. Note that the server-side DISCONNECT is a new feature of MQTT v5 (in MQTT v3.1.1, the server needed to drop the connection). RS MUST stop forwarding messages to the unauthorized subscribers.

In the case of a PUBACK with '0x87 (Not authorized)', the client can update its token using the Re-authentication feature of MQTT v5.

Also, the clients can proactively update their tokens without waiting for such a PUBACK. To re-authenticate, the client sends an AUTH packet with reason code '0x19 (Re-authentication)'. The client MUST set the authentication method as 'ace' and transport the new token in the Authentication Data. The client and the RS go through the same steps for proof of possession validation as described in the previous section. If the re-authentication fails, the server MUST send a DISCONNECT with the reason code '0x87 (Not Authorized)'.

4. IANA Considerations

The following registrations are done for the ACE OAuth Profile Registry following the procedure specified in [I-D.ietf-ace-oauth-authz].

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

Profile name: mqtt_tls

Profile description: Profile for delegating client authentication and authorization using MQTT as the application protocol and TLS For transport layer security.

Profile ID:

Change controller: IESG

Reference: [RFC-XXXX]

5. Security Considerations

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

In addition, the security considerations outlined in MQTT v3.1.1 - the OASIS Standard [MQTT-OASIS-Standard] and MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5] apply. Mainly, this document provides an authorization solution for MQTT, the responsibility of which is left to the specific implementation in MQTT v5 - the OASIS Standard [MQTT-OASIS-Standard-v5]. In the following, we comment on a few relevant issues based on the current MQTT specifications.

In this document, RS uses the PoP access token to authenticate the client. If the client is able, TLS certificates sent from the client

can be used by the RS to authenticate the client. The TLS certificate from the RS MUST be used by the client to authenticate the RS.

To authorize a client's publish and subscribe requests in an ongoing session, the RS caches the access token after accepting the connection from the client. However, if some permissions are revoked in the meantime, the RS may still grant publish/subscribe to revoked topics until the session ends or the token expires. When permissions change dynamically, it is expected that AS follows a reasonable expiration strategy for the access tokens.

The RS may monitor client behaviour to detect potential security problems, especially those affecting availability. These include repeated token transfer attempts to the public "authz-info" topic, repeated connection attempts, abnormal terminations, and clients that connect but do not send any data. If the RS supports the public "authz-info" topic, described in Appendix B, then this may be vulnerable to a DDoS attack, where many clients use the "authz-info" public topic to transport fictitious tokens, which RS may need to store indefinitely.

6. Privacy Considerations

The privacy considerations outlined in [I-D.ietf-ace-oauth-authz] apply to this work.

In MQTT, the RS is a central trusted party and may forward potentially sensitive information between clients. Clients may choose to encrypt the payload of their messages. However, this would not provide privacy for other properties of the message such as topic name.

7. References

7.1. Normative References

[I-D.gerdes-ace-dtls-authorize]

Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-gerdes-ace-dtls-authorize-01 (work in progress), March 2017.

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[MQTT-OASIS-Standard]

Banks, A., Ed. and R. Gupta, Ed., "OASIS Standard MQTT Version 3.1.1 Plus Errata 01", 2015, <<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>>.

[MQTT-OASIS-Standard-v5]

Banks, A., Ed., Briggs, E., Ed., Borgendale, K., Ed., and R. Gupta, Ed., "OASIS Standard MQTT Version 5.0", 2017, <<http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

[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>>.

7.2. Informative References

[fremantle14]

Fremantle, P., Aziz, B., Kopecky, J., and P. Scott, "Federated Identity and Access Management for the Internet of Things", research International Workshop on Secure Internet of Things, September 2014, <<http://dx.doi.org/10.1109/SIoT.2014.8>>.

[I-D.ietf-ace-actors]

Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-07 (work in progress), October 2018.

- [I-D.ietf-ace-cwt-proof-of-possession]
Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-06 (work in progress), February 2019.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <<https://www.rfc-editor.org/info/rfc7800>>.

Appendix A. Checklist for profile requirements

- o AS discovery: For the basic protocol using either MQTT v3.1.1 or MQTT v5, the clients/client authorization servers need to be configured out-of-band. RS does not provide any hints to help AS discovery. AS discovery is possible with the MQTT v5 extensions described in Section 3.
- o The communication protocol between the client and RS: MQTT
- o The security protocol between the client and RS: TLS
- o Client and RS mutual authentication: RS provides a server certificate during TLS handshake. Client transports token and MAC via the MQTT CONNECT message. Other methods for transporting the token with the MQTT v5 extensions described in Section 3.
- o Content format: For the HTTPS interactions with AS, "application/json". The MQTT payloads may be formatted JSON or CBOR.
- o PoP protocols: Either symmetric or asymmetric keys can be supported.
- o Unique profile identifier: mqtt_tls
- o Token introspection: RS uses HTTPS /introspect interface of AS.
- o Token request: CAS uses HTTPS /token interface of AS.

- o /authz-info endpoint: It MAY be supported using the method described in Appendix B, not protected.
- o Token transport: In MQTT CONNECT message or using the AUTH extensions for MQTT v5 described in Section 3.

Appendix B. The Authorization Information Endpoint

The main document described a method for transporting tokens inside MQTT CONNECT messages. In this section, we describe an alternative method to transport an access token.

The method consists of the MQTT broker accepting PUBLISH messages to a public "authz-info" topic. A client using this method MUST first connect to the broker, and publish the access token using the "authz-info" topic. The broker must verify the validity of the token (i.e., through local validation or introspection). After publishing the token, the client disconnects from the broker and is expected to try reconnecting over TLS.

In MQTT v3.1.1, after the client published to the "authz-info" topic, it is not possible for the broker to communicate the result of the token verification. In MQTT v5, the broker can return 'Not authorized' error to a PUBLISH request for QoS greater or equal to 1. In any case, any token authorization failure affect the subsequent TLS handshake, which can prompt the client to obtain a valid token.

Appendix C. Document Updates

Version 01 updates Version 00 as follows:

- o Adds Section 3 to describe improvements to the basic protocol operation with the new MQTT v5 - OASIS Committee Specification, including improved authentication exchange and error reporting.
- o Condenses background information specific to MQTT in Section 2.
- o Clarifies token transport and token structure in Section 2.1.2 and Section 2.1.3.
- o Removes Appendix on error reporting as this is now handled with MQTT v5.

Version 02 updates Version 01 as follows:

- o Adds PINGREQ packet for token expiry checks.
- o Minor typo fixes.

Version 03 updates Version 02 as follows:

- o Requirements language fixed according to the new IETF recommendation.
- o The use of audience and scopes claims in access tokens has been clarified.
- o Encoding used in the access token has been clarified.
- o Sections on IANA, security and privacy considerations are added.

Version 04 updates Version 03 as follows:

- o Simplified protocol exchanges (e.g., eliminated alternatives) and added clarifications (e.g., PoP in CONNECT)
- o Updated references to the ACE framework document

Acknowledgements

The authors would like to thank Ludwig Seitz for his review and his input on the authorization information endpoint, presented in the appendix.

Authors' Addresses

Cigdem Sengul
Nominet
2 Kingdom Street
London W2 6BD
UK

Email: Cigdem.Sengul@nominet.uk

Anthony Kirby
Oxbotica
1a Milford House, Mayfield Road, Summertown
Oxford OX2 7EL
UK

Email: anthony@anthony.org

Paul Fremantle
University of Portsmouth
School of Computing, Buckingham House
Portsmouth PO1 3HE
UK

Email: paul.fremantle@port.ac.uk