ACE   P. van der Stok
Internet-Draft   Consultant
Intended status: Standards Track   P. Kampanakis
Expires: July 9, 2020   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.

Table of Contents

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 inaccurracies 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.
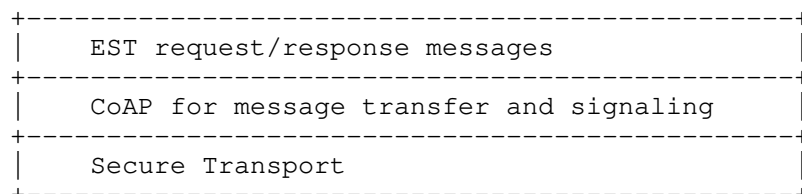
```
+-------------------------------------------------+
|      EST request/response messages              |
+-------------------------------------------------+
|      CoAP for message transfer and signaling    |
+-------------------------------------------------+
|      Secure Transport                           |
+-------------------------------------------------+
```

             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 /crts 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
immediatelly following a /crts 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       |   /crts                     |
|   /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 /crts, /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 (/crts, /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
/crts, /att	2.05	Success. Certs included in the response payload.
	4.xx / 5.xx	Failure.
/sen, /skg, /sren, /skc	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 cacerts 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 serer-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.

```
                                       Constrained Network
    .------.                    .--------------------------.
    |  CA  |                    | .------------------------. |
    '_____'                    | |                        | |
       |                        | |                        | |
    .------.  HTTP    .----------------.  CoAPS   .----------. | |
    | EST  |<------->|EST-coaps-to-HTTPS|<------->| EST Client| |
    |Server|over TLS |   Registrar     |         '----------' | |
    '_____'         '_____'          | |          | |
                                      | |                      | |
                                      | '----------------------' |
                                      '--------------------------'
```
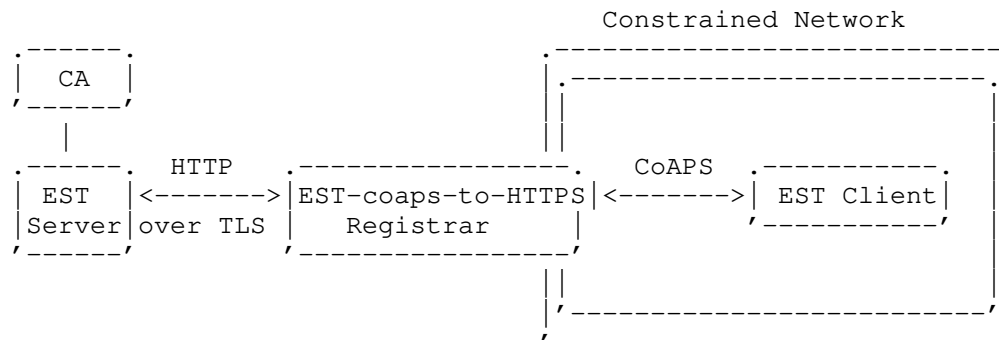
            Figure 4: EST-coaps-to-HTTPS Registrar at the CoAP boundary.

   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;      |  280  | [RFC7030] [I-D.ietf-lamps-|
| smime-type=server-generated- |       | rfc5751-bis] [ThisRFC]   |
| key                          |       |                          |
| application/pkcs7-mime;      |  281  | [I-D.ietf-lamps-rfc5751-bi|
| smime-type=certs-only        |       | s] [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        | TBD28 |  [RFC2585] [ThisRFC]      |
|                              |   7   |                          |
+------------------------------+-------+--------------------------+
```

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.crts". 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	Change	Referen	Status	Related	Date Re	Date
Suff	Contro	ces		Informat	gistere	Modified
ix	ller			ion	d	
   +------+--------+---------+---------+----------+---------+----------+
est	IETF	[RFC703	permane		2013-08	[THIS
		0]	nt		-16	RFC's pu
		[THIS				blicatio
		RFC]				n 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 successfuly 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 [tripleshake] 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 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 Furuhed 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
3131323730335a3067310b300906035504061302555533310b3009060355
04080c024341310b300906035504070c024c4131143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3110300e06035504030c07526f6f7420434130593013060072a86
48ce3d020106082a8648ce3d030107034200040c1b1e82ba8cc72680973f
97edb8a0c72ab0d405f05d4fe29b997a14ccce89008313d09666b6ce375c
595fcc8e37f8e4354497011be90e56794bd91ad951ab45a3818430818130
1d0603551d0e041604141df1208944d77b5f1d9dcb51ee244a523f3ef5de
301f0603551d230418301680141df1208944d77b5f1d9dcb51ee244a523f
3ef5de300f0603551d130101ff040530030101ff300e0603551d0f0101ff
040403020106301e0603551d11041730158113636572746966679406578 61
6d706c652e636f6d300a06082a8648ce3d0403020348003045022 02b891d
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
300d06035504051306577431323334305930130607 2a8648ce3d02010608
2a8648ce3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f
028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75
f602f9152618f816a2b23b5638e59fd9a073303406092a864886f70d0109
0731270c2576437630292a264a4b4a3bc3a2c280c2992f3e3c2e2c3d6b6e
7634332323403d204e787e60303b06092a864886f70d01090e312e302c30
2a0603551d1104233021a01f06082b06010505070804a013301106092b06
010401b43b0a01040401020304300a06082a8648ce3d0403020348003045
02210092563a546463bd9ecff170d0fd1f2ef0d3d012160e5ee90cffedab
ec9b9a38920220179f10a3436109051abad17590a09bc87c4dce5453a6fc
1135a1e84eed754377
```

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
78616d706c6520496e6331163014060355040b0c0d63657274696669636 1
74696f6e3113301106035504030c0a3830322e31415220434130201 70d31
3930313331313131323931365a180f39393939393132333313233353935395a30
5c310b300906035504061302555531 0b300906035504080c024341310b30
0906035504070c024c413114301206 0355040a0c0b6578616d706c652049
6e63310c300a060355040b0c03496f54310f300d0603550405130657 7431
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
79295398774b2ff8e7e82abb0c17eaef344a5088fa69fd63ee611850c34b
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)
30818702010030130607 2a8648ce3d020106082a8648ce3d030107046d30
6b020101042061336a86ac6e7af4a96f632830ad4e6aa0837679206094d7
679a01ca8c6f0c37a14403420004c8b421f11c25e47e3ac57123bf2d9fdc
494f028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95
cf75f602f9152618f816a2b23b5638e59fd9
19 0119                                   # unsigned(281)
59 01D3                                   # bytes(467)
308201cf06092a864886f70d010702a08201c0308201bc0201013100300b
06092a864886f70d010701a08201a23082019e30820144a0030201020209
00b3313e8f3fc9538e300a06082a8648ce3d0403023016311430120603 55
040a0c0b736b67206578616d706c65301e170d3139303039303034303734343 0
335a170d3339303833303037343430335a3016311430120603 55040a0c0b
736b67206578616d706c65305930130607 2a8648ce3d020106082a8648ce
3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351
cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75f602f915
2618f816a2b23b5638e59fd9a37b307930090603551d1304023000302c06
096086480186f842010d041f161d4f70656e53534c2047656e6572617465
6420436572746966696361746530 1d0603551d0e0416041496600d8716bf
7fd0e752d0ac760777ad665d02a0301f0603551d23041830 16801496600d
8716bf7fd0e752d0ac760777ad665d02a0300a06082a8648ce3d04030203
48003045022100e95bfa25a08976652246f2d96143da39fce0dc4c9b26b9
cce1f24164cc2b12b602201351fd8eea65764e3459d324e4345ff5b2a915
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.]

   307c06072b0601010101163022060388370131
1b131950617273652053455
   420617320322e3939392e31206461746106092a864886f70d010907302c06
   03883702312506038837030603883704131950617273652053455420617
32
   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 exmple, 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**(SZX+4)) which gives 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

          Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)
                 draft-ietf-ace-cwt-proof-of-possession-11

Abstract

   This specification describes how to declare in a CBOR Web Token (CWT)
   (which is defined by RFC 8392) that the presenter of the CWT
   possesses a particular proof-of-possession key. Being able to prove
   possession of a key is also sometimes described as being the holder-
   of-key. This specification provides equivalent functionality to
   "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" (RFC
   7800) but using Concise Binary Object Representation (CBOR) and CWTs
   rather than JavaScript Object Notation (JSON) and JSON Web Tokens
   (JWTs).

Status of This Memo

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

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

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

   This Internet-Draft will expire on May 3, 2020.

Copyright Notice

Table of Contents

1. Introduction

   This specification describes how a CBOR Web Token (CWT) [RFC8392] can
   declare that the presenter of the CWT possesses a particular proof-
   of-possession (PoP) key. Proof of possession of a key is also
   sometimes described as being the holder-of-key. This specification

provides equivalent functionality to "Proof-of-Possession Key
Semantics for JSON Web Tokens (JWTs)" [RFC7800] but using Concise
Binary Object Representation (CBOR) [RFC7049] and CWTs [RFC8392]
rather than JavaScript Object Notation (JSON) [RFC8259] and JSON Web
Tokens (JWTs) [JWT].

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.

   This specification uses terms defined in the CBOR Web Token (CWT)
   [RFC8392], CBOR Object Signing and Encryption (COSE) [RFC8152], and
   Concise Binary Object Representation (CBOR) [RFC7049] specifications.

   These terms are defined by this specification:

   Issuer
      Party that creates the CWT and binds the claims about the subject
      to the proof-of-possession key.

   Presenter
      Party that proves possession of a private key (for asymmetric key
      cryptography) or secret key (for symmetric key cryptography) to a
      recipient of a CWT.
      In the context of OAuth, this party is also called the OAuth
      Client.

   Recipient
      Party that receives the CWT containing the proof-of-possession key
      information from the presenter.
      In the context of OAuth, this party is also called the OAuth
      Resource Server.

   This specification provides examples in CBOR extended diagnostic
   notation, as defined in Appendix G of [RFC8610]. The examples
   include line breaks for readability.

3. Representations for Proof-of-Possession Keys

   By including a "cnf" (confirmation) claim in a CWT, the issuer of the
   CWT declares that the presenter possesses a particular key and that
   the recipient can cryptographically confirm that the presenter has
   possession of that key. The value of the "cnf" claim is a CBOR map

(which is defined in Section 2.1 of [RFC7049]) and the members of
that map identify the proof-of-possession key.

The presenter can be identified in one of several ways by the CWT,
depending upon the application requirements. For instance, some
applications may use the CWT "sub" (subject) claim [RFC8392], to
identify the presenter. Other applications may use the "iss"
(issuer) claim [RFC8392] to identify the presenter. In some
applications, the subject identifier might be relative to the issuer
identified by the "iss" claim. The actual mechanism used is
dependent upon the application. The case in which the presenter is
the subject of the CWT is analogous to Security Assertion Markup
Language (SAML) 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation
usage.

3.1. Confirmation Claim

The "cnf" claim in the CWT is used to carry confirmation methods.
Some of them use proof-of-possession keys while others do not. This
design is analogous to the SAML 2.0 [OASIS.saml-core-2.0-os]
SubjectConfirmation element in which a number of different subject
confirmation methods can be included (including proof-of-possession
key information).

The set of confirmation members that a CWT must contain to be
considered valid is context dependent and is outside the scope of
this specification. Specific applications of CWTs will require
implementations to understand and process some confirmation members
in particular ways. However, in the absence of such requirements,
all confirmation members that are not understood by implementations
MUST be ignored.

This specification establishes the IANA "CWT Confirmation Methods"
registry for these members in Section 7.2 and registers the members
defined by this specification. Other specifications can register
other members used for confirmation, including other members for
conveying proof-of-possession keys using different key
representations.

The "cnf" claim value MUST represent only a single proof-of-
possession key. At most one of the "COSE_Key" and
"Encrypted_COSE_Key" confirmation values defined in Figure 1 may be
present. Note that if an application needs to represent multiple
proof-of-possession keys in the same CWT, one way for it to achieve
this is to use other claim names, in addition to "cnf", to hold the
additional proof-of-possession key information. These claims could
use the same syntax and semantics as the "cnf" claim. Those claims
would be defined by applications or other specifications and could be

registered in the IANA "CBOR Web Token Claims" registry
[IANA.CWT.Claims].

```
/-------------------+-----+-----------------------------\
| Name              | Key | Value type                  |
|-------------------+-----+-----------------------------|
| COSE_Key          | 1   | COSE_Key                    |
| Encrypted_COSE_Key | 2  | COSE_Encrypt or COSE_Encrypt0 |
| kid               | 3   | binary string               |
\-------------------+-----+-----------------------------/
```

        Figure 1: Summary of the cnf names, keys, and value types

3.2. Representation of an Asymmetric Proof-of-Possession Key

   When the key held by the presenter is an asymmetric private key, the
   "COSE_Key" member is a COSE_Key [RFC8152] representing the
   corresponding asymmetric public key. The following example
   demonstrates such a declaration in the CWT Claims Set of a CWT:

```
     {
    /iss/ 1 : "coaps://server.example.com",
    /aud/ 3 : "coaps://client.example.org",
    /exp/ 4 : 1879067471,
    /cnf/ 8 :{
      /COSE_Key/ 1 :{
        /kty/ 1 : /EC2/ 2,
        /crv/ -1 : /P-256/ 1,
        /x/ -2 : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa27c9
                  e354089bbe13',
        /y/ -3 : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020731e
                  79a3b4e47120'
      }
    }
   }
```

   The COSE_Key MUST contain the required key members for a COSE_Key of
   that key type and MAY contain other COSE_Key members, including the
   "kid" (Key ID) member.

   The "COSE_Key" member MAY also be used for a COSE_Key representing a
   symmetric key, provided that the CWT is encrypted so that the key is
   not revealed to unintended parties. The means of encrypting a CWT is
   explained in [RFC8392]. If the CWT is not encrypted, the symmetric
   key MUST be encrypted as described in Section 3.3. This procedure is
   equivalent to the one defined in section 3.3 of [RFC7800].

3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key

   When the key held by the presenter is a symmetric key, the
   "Encrypted_COSE_Key" member is an encrypted COSE_Key [RFC8152]
   representing the symmetric key encrypted to a key known to the
   recipient using COSE_Encrypt or COSE_Encrypt0.

   The following example illustrates a symmetric key that could
   subsequently be encrypted for use in the "Encrypted_COSE_Key" member:

```
     {
      /kty/ 1 : /Symmetric/ 4,
      /alg/ 3 : /HMAC 256-256/ 5,
      /k/ -1 : h'6684523ab17337f173500e5728c628547cb37df
                 e68449c65f885d1b73b49eae1'
     }
```

   The COSE_Key representation is used as the plaintext when encrypting
   the key.

   The following example CWT Claims Set of a CWT illustrates the use of
   an encrypted symmetric key as the "Encrypted_COSE_Key" member value:

```
  {
   /iss/ 1 : "coaps://server.example.com",
   /sub/ 2 : "24400320",
   /aud/ 3: "s6BhdRkqt3",
   /exp/ 4 : 1311281970,
   /iat/ 5 : 1311280970,
   /cnf/ 8 : {
   /Encrypted_COSE_Key/ 2 : [
       /protected header/ h'A1010A' /{ \alg\ 1:10 \AES-CCM-16-64-128\}/,
       /unprotected header/ { / iv / 5: h'636898994FF0EC7BFCF6D3F95B'},
       /ciphertext/  h'0573318A3573EB983E55A7C2F06CADD0796C9E584F1D0E3E
                       A8C5B052592A8B2694BE9654F0431F38D5BBC8049FA7F13F'
     ]
    }
  }
```

   The example above was generated with the key:

```
          h'6162630405060708090a0b0c0d0e0f10'
```

3.4. Representation of a Key ID for a Proof-of-Possession Key

   The proof-of-possession key can also be identified using a Key ID
   instead of communicating the actual key, provided the recipient is
   able to obtain the identified key using the Key ID. In this case,
   the issuer of a CWT declares that the presenter possesses a
   particular key and that the recipient can cryptographically confirm
   proof of possession of the key by the presenter by including a "cnf"
   claim in the CWT whose value is a CBOR map with the CBOR map
   containing a "kid" member identifying the key.

   The following example demonstrates such a declaration in the CWT
   Claims Set of a CWT:

```
  {
   /iss/ 1 : "coaps://as.example.com",
   /aud/ 3 : "coaps://resource.example.org",
   /exp/ 4 : 1361398824,
   /cnf/ 8 : {
     /kid/ 3 : h'dfd1aa976d8d4575a0fe34b96de2bfad'
    }
  }
```

   The content of the "kid" value is application specific. For
   instance, some applications may choose to use a cryptographic hash of
   the public key value as the "kid" value.

   Note that the use of a Key ID to identify a proof-of-possession key
   needs to be carefully circumscribed, as described below and in
   Section 6. In cases where the Key ID is not a cryptographic value
   derived from the key or where not all of the parties involved are
   validating the cryptographic derivation, implementers should expect
   collisions, where different keys are assigned the same Key ID.
   Recipients of a CWT with a PoP key linked through only a Key ID
   should be prepared to handle such situations.

   In the world of constrained Internet of Things (IoT) devices, there
   is frequently a restriction on the size of Key IDs, either because of
   table constraints or a desire to keep message sizes small.

   Note that the value of a Key ID for a specific key is not necessarily
   the same for different parties. When sending a COSE encrypted
   message with a shared key, the Key ID may be different on both sides
   of the conversation, with the appropriate one being included in the
   message based on the recipient of the message.

3.5. Specifics Intentionally Not Specified

   Proof of possession is often demonstrated by having the presenter
   sign a value determined by the recipient using the key possessed by
   the presenter. This value is sometimes called a "nonce" or a
   "challenge". There are, however, also other means to demonstrate
   freshness of the exchange and to link the proof-of-possession key to
   the participating parties, as demonstrated by various authentication
   and key exchange protocols.

   The means of communicating the nonce and the nature of its contents
   are intentionally not described in this specification, as different
   protocols will communicate this information in different ways.
   Likewise, the means of communicating the signed nonce is also not
   specified, as this is also protocol specific.

   Note that other means of proving possession of the key exist, which
   could be used in conjunction with a CWT's confirmation key.
   Applications making use of such alternate means are encouraged to
   register them in the IANA "CWT Confirmation Methods" registry
   established in Section 7.2.

4. Security Considerations

   All the security considerations that are discussed in [RFC8392] also
   apply here. In addition, proof of possession introduces its own
   unique security issues. Possessing a key is only valuable if it is
   kept secret. Appropriate means must be used to ensure that
   unintended parties do not learn private key or symmetric key values.

   Applications utilizing proof of possession SHOULD also utilize
   audience restriction, as described in Section 3.1.3 of [RFC8392], as
   it provides additional protections. Audience restriction can be used
   by recipients to reject messages intended for different recipients.
   (Of course, applications not using proof of possession can also
   benefit from using audience restriction to reject messages intended
   for different recipients.)

   CBOR Web Tokens with proof-of-possession keys are used in context of
   an architecture, such as the ACE OAuth Framework
   [I-D.ietf-ace-oauth-authz], in which protocols are used by a
   presenter to request these tokens and to subsequently use them with
   recipients. Proof of possession only provides the intended security
   gains when the proof is known to be current and not subject to replay
   attacks; security protocols using mechanisms such as nonces and
   timestamps can be used to avoid the risk of replay when performing
   proof of possession for a token. Note that a discussion of the

architecture or specific protocols that CWT proof-of-possession
tokens are used with is beyond the scope of this specification.

As is the case with other information included in a CWT, it is
necessary to apply data origin authentication and integrity
protection (via a keyed message digest or a digital signature). Data
origin authentication ensures that the recipient of the CWT learns
about the entity that created the CWT since this will be important
for any policy decisions. Integrity protection prevents an adversary
from changing any elements conveyed within the CWT payload. Special
care has to be applied when carrying symmetric keys inside the CWT
since those not only require integrity protection but also
confidentiality protection.

As described in Section 6 (Key Identification) and Appendix D (Notes
on Key Selection) of [JWS], it is important to make explicit trust
decisions about the keys. Proof-of-possession signatures made with
keys not meeting the application's trust criteria MUST NOT be relied
upon.

5. Privacy Considerations

A proof-of-possession key can be used as a correlation handle if the
same key is used on multiple occasions. Thus, for privacy reasons,
it is recommended that different proof-of-possession keys be used
when interacting with different parties.

6. Operational Considerations

The use of CWTs with proof-of-possession keys requires additional
information to be shared between the involved parties in order to
ensure correct processing. The recipient needs to be able to use
credentials to verify the authenticity and integrity of the CWT.
Furthermore, the recipient may need to be able to decrypt either the
whole CWT or the encrypted parts thereof (see Section 3.3). This
requires the recipient to know information about the issuer.
Likewise, there needs to be agreement between the issuer and the
recipient about the claims being used (which is also true of CWTs in
general).

When an issuer creates a CWT containing a Key ID claim, it needs to
make sure that it does not issue another CWT with different claims
containing the same Key ID within the lifetime of the CWTs, unless
intentionally desired. Failure to do so may allow one party to
impersonate another party, with the potential to gain additional
privileges. A case where such reuse of a Key ID would be intentional
is when a presenter obtains a CWT with different claims (e.g.,
extended scope) for the same recipient, but wants to continue using

an existing security association (e.g., a DTLS session) bound to the
key identified by the Key ID. Likewise, if PoP keys are used for
multiple different kinds of CWTs in an application and the PoP keys
are identified by Key IDs, care must be taken to keep the keys for
the different kinds of CWTs segregated so that an attacker cannot
cause the wrong PoP key to be used by using a valid Key ID for the
wrong kind of CWT. Using an audience restriction for the CWT would
be one strategy to mitigate this risk.

7. IANA Considerations

   The following registration procedure is used for all the registries
   established by this specification.

   Values are registered on a Specification Required [RFC8126] basis
   after a three-week review period on the cwt-reg-review@ietf.org
   mailing list, on the advice of one or more Designated Experts.
   However, to allow for the allocation of values prior to publication,
   the Designated Experts may approve registration once they are
   satisfied that such a specification will be published. [[Note to
   the RFC Editor: The name of the mailing list should be determined in
   consultation with the IESG and IANA. Suggested name: cwt-reg-
   review@ietf.org.]]

   Registration requests sent to the mailing list for review should use
   an appropriate subject (e.g., "Request to Register CWT Confirmation
   Method: example"). Registration requests that are undetermined for a
   period longer than 21 days can be brought directly to IANA's
   attention (using the iana@iana.org mailing list) for resolution.

   Designated Experts should determine whether a registration request
   contains enough information for the registry to be populated with the
   new values and whether the proposed new functionality already exists.
   In the case of an incomplete registration or an attempt to register
   already existing functionality, the Designated Experts should ask for
   corrections or reject the registration.

   It is suggested that multiple Designated Experts be appointed who are
   able to represent the perspectives of different applications using
   this specification in order to enable broadly informed review of
   registration decisions. In cases where a registration decision could
   be perceived as creating a conflict of interest for a particular
   Expert, that Expert should defer to the judgment of the other
   Experts.

7.1. CBOR Web Token Claims Registration

   This specification registers the "cnf" claim in the IANA "CBOR Web
   Token Claims" registry [IANA.CWT.Claims] established by [RFC8392].

7.1.1. Registry Contents

   o Claim Name: "cnf"
   o Claim Description: Confirmation
   o JWT Claim Name: "cnf"
   o Claim Key: TBD (maybe 8)
   o Claim Value Type(s): map
   o Change Controller: IESG
   o Specification Document(s): Section 3.1 of [[this document]]

7.2. CWT Confirmation Methods Registry

   This specification establishes the IANA "CWT Confirmation Methods"
   registry for CWT "cnf" member values. The registry records the
   confirmation method member and a reference to the specification that
   defines it.

7.2.1. Registration Template

   Confirmation Method Name:
      The human-readable name requested (e.g., "kid").

   Confirmation Method Description:
      Brief description of the confirmation method (e.g., "Key
      Identifier").

   JWT Confirmation Method Name:
      Claim Name of the equivalent JWT confirmation method value, as
      registered in [IANA.JWT.Claims]. CWT claims should normally have
      a corresponding JWT claim. If a corresponding JWT claim would not
      make sense, the Designated Experts can choose to accept
      registrations for which the JWT Claim Name is listed as "N/A".

   Confirmation Key:
      CBOR map key value for the confirmation method.

   Confirmation Value Type(s):
      CBOR types that can be used for the confirmation method value.

   Change Controller:
      For Standards Track RFCs, list the "IESG". For others, give the
      name of the responsible party.

      Specification Document(s):
         Reference to the document or documents that specify the parameter,
         preferably including URIs that can be used to retrieve copies of
         the documents. An indication of the relevant sections may also be
         included but is not required. Note that the Designated Experts
         and IANA must be able to obtain copies of the specification
         document(s) to perform their work.

7.2.2. Initial Registry Contents

      o Confirmation Method Name: "COSE_Key"
      o Confirmation Method Description: COSE_Key Representing Public Key
      o JWT Confirmation Method Name: "jwk"
      o Confirmation Key: 1
      o Confirmation Value Type(s): COSE_Key structure
      o Change Controller: IESG
      o Specification Document(s): Section 3.2 of [[this document]]

      o Confirmation Method Name: "Encrypted_COSE_Key"
      o Confirmation Method Description: Encrypted COSE_Key
      o JWT Confirmation Method Name: "jwe"
      o Confirmation Key: 2
      o Confirmation Value Type(s): COSE_Encrypt or COSE_Encrypt0
         structure (with an optional corresponding COSE_Encrypt or
         COSE_Encrypt0 tag)
      o Change Controller: IESG
      o Specification Document(s): Section 3.3 of [[this document]]

      o Confirmation Method Name: "kid"
      o Confirmation Method Description: Key Identifier
      o JWT Confirmation Method Name: "kid"
      o Confirmation Key: 3
      o Confirmation Value Type(s): binary string
      o Change Controller: IESG
      o Specification Document(s): Section 3.4 of [[this document]]

8. References

8.1. Normative References

   [IANA.CWT.Claims]
              IANA, "CBOR Web Token Claims",
              <http://www.iana.org/assignments/cwt>.

   [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>.

   [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
              Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
              October 2013, <https://www.rfc-editor.org/info/rfc7049>.

   [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>.

8.2. Informative 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)", draft-ietf-ace-oauth-authz-21
              (work in progress), February 2019.

   [IANA.JWT.Claims]
              IANA, "JSON Web Token Claims",
              <http://www.iana.org/assignments/jwt>.

   [JWS]   Jones, M., Bradley, J., and N. Sakimura, "JSON Web
              Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May
              2015, <http://www.rfc-editor.org/info/rfc7515>.

   [JWT]   Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
              (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,
              <http://www.rfc-editor.org/info/rfc7519>.

   [OASIS.saml-core-2.0-os]
              Cantor, S., Kemp, J., Philpott, R., and E. Maler,
              "Assertions and Protocol for the OASIS Security Assertion
              Markup Language (SAML) V2.0", OASIS Standard saml-core-
              2.0-os, March 2005,
              <http://docs.oasis-open.org/security/saml/v2.0/>.

   [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>.

   [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>.

   [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>.

Acknowledgements

Document History

   [[to be removed by the RFC Editor before publication as an RFC]]

   -11

   o Addressed remaining IESG review comment by Mirja Kuehlewind.

   -10

   o Addressed IESG review comments by Adam Roach and Eric Vyncke.

   -09

   o Addressed Gen-ART review comments by Christer Holmberg and SecDir
      review comments by Yoav Nir.

   -08

   o Addressed remaining Area Director review comments by Benjamin
      Kaduk.

-07

o Addressed Area Director review by Benjamin Kaduk.

-06

o Corrected nits identified by Roman Danyliw.

-05

o Added text suggested by Jim Schaad describing considerations when
   using the Key ID confirmation method.

-04

o Addressed additional WGLC comments by Jim Schaad and Roman
   Danyliw.

-03

o Addressed review comments by Jim Schaad, see https://www.ietf.org/
   mail-archive/web/ace/current/msg02798.html

o Removed unnecessary sentence in the introduction regarding the use
   any strings that could be case-sensitive.

o Clarified the terms Presenter and Recipient.

o Clarified text about the confirmation claim.

-02

o Changed "typically" to "often" when describing ways of performing
   proof of possession.

o Changed b64 to hex encoding in an example.

o Changed to using the RFC 8174 boilerplate instead of the RFC 2119
   boilerplate.

-01

o Now uses CBOR diagnostic notation for the examples.

o Added a table summarizing the "cnf" names, keys, and value types.

o Addressed some of Jim Schaad's feedback on -00.

   -00

   o Created the initial working group draft from draft-jones-ace-cwt-
      proof-of-possession-01.

Authors' Addresses

   Michael B. Jones
   Microsoft

   Email: mbj@microsoft.com
   URI: http://self-issued.info/

   Ludwig Seitz
   RISE SICS
   Scheelevaegen 17
   Lund 223 70
   Sweden

   Email: ludwig@ri.se

   Goeran Selander
   Ericsson AB
   Faeroegatan 6
   Kista 164 80
   Sweden

   Email: goran.selander@ericsson.com

   Samuel Erdtman
   Spotify

   Email: erdtman@spotify.com

   Hannes Tschofenig
   Arm Ltd.
   Hall in Tirol 6060
   Austria

   Email: Hannes.Tschofenig@arm.com

ACE Working Group   S. Gerdes
Internet-Draft   O. Bergmann
Intended status: Standards Track   C. Bormann
Expires: May 2, 2021   Universitaet Bremen TZI
                                                             G. Selander
                                                            Ericsson AB
                                                               L. Seitz
                                                              Combitech
                                                       October 29, 2020

Datagram Transport Layer Security (DTLS) Profile for Authentication and
          Authorization for Constrained Environments (ACE)
                   draft-ietf-ace-dtls-authorize-14

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 May 2, 2021.

Copyright Notice

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

   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. 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].

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 5.8.1 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):

```
    C                             RS                        AS
    | [---- Resource Request ------>]|                       |
    |                                |                       |
    | [<-AS Request Creation Hints-] |                       |
    |                                |                       |
    | ------- Token Request  -------------------------------> |
    |                                |                       |
    | <------------------------- Access Token --------- |
    |                                + Access Information   |
```
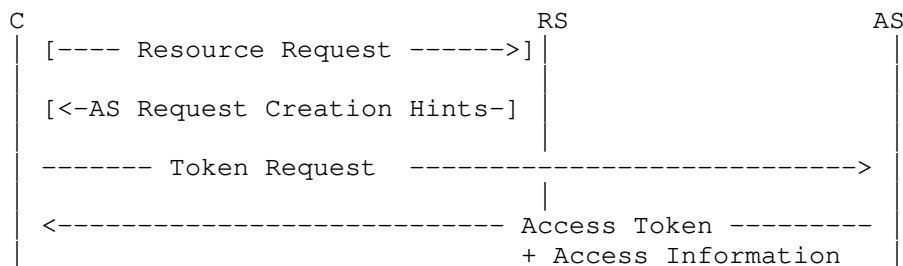
Figure 1: Retrieving an Access Token

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 5.1.2 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 may have been registered at the authorization server via the
OAuth 2.0 client registration mechanism as outlined in Section 5.3 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 5.8.1 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.

```
   C                              RS                          AS
   | [--- Access Token ------>] |                            |
   |                            |                            |
   | <== DTLS channel setup ==> |                            |
   |                            |                            |
   | == Authorized Request ===> |                            |
   |                            |                            |
   | <=== Protected Resource == |                            |
```
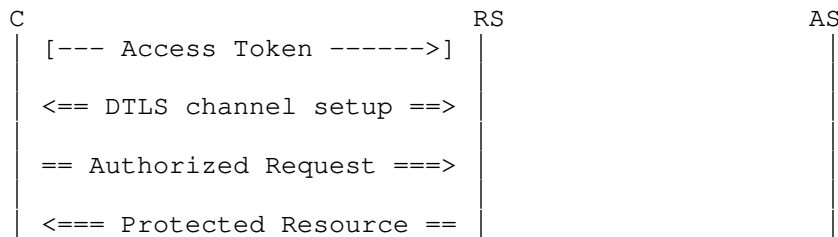
                      Figure 2: Protocol overview

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 (see Section 5.6 of
   [I-D.ietf-ace-oauth-authz]) 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:

   o The client MUST securely have obtained keying material to
      communicate with the authorization server.

   o 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.

   o 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. RawPublicKey Mode

   When the client uses RawPublicKey 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,
        req_aud    : "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 MAY contain a "profile" parameter with the value
"coap_dtls" to indicate that this profile MUST be used for
communication between the client and the resource server. The
"profile" may be specified out-of-band, in which case it does not
have to be sent. 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.
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 SHOULD proceed 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].

   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 ciphersuite
   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. This specification
   therefore mandates implementation 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. 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
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 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. 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 5.6.3. 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:

o OKM, the output keying material, is the derived symmetric key

o salt is the empty byte string

o IKM, the input keying material, is the key derivation key as
   defined above

o info is the serialization of a CBOR array consisting of
   ([RFC8610]):

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

where:

o type is set to the constant text string "ACE-CoAP-DTLS-key-
   derivation",

o L is the size of the symmetric key in bytes,

o 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
[I-D.ietf-cbor-7049bis]. 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 additionally 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 "exi" 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 is
expected to offer at least the ciphersuite TLS_PSK_WITH_AES_128_CCM_8
[RFC6655] to the resource server.

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
that 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

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:

o 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.

o 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 resource
server uses the "COSE_KDF_Context" information 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 5.8.1 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 5.8.3 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 5.1.1 of [I-D.ietf-ace-oauth-authz].

The resource server MUST only accept an incoming CoAP request as
authorized if the following holds:

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 5.8.2 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 5.1.2 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 5.6.3 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.

```
    C                              RS                     AS
    | <===== DTLS channel =====> |                        |
    |         + Access Token     |                        |
    |                            |                        |
    | --- Token Request  ------------------------------>  |
    |                            |                        |
    | <------------------------- New Access Token -       |
    |                              + Access Information    |
    |                            |                        |
    | --- Update /authz-info --> |                        |
    |      New Access Token      |                        |
    |                            |                        |
    | == Authorized Request ===> |                        |
    |                            |                        |
    | <=== Protected Resource == |                        |
    |                            |                        |
```
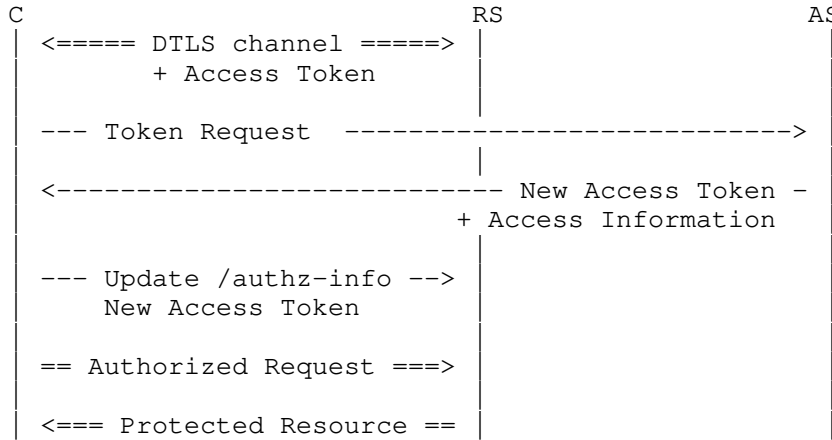
              Figure 10: Overview of Dynamic Update Operation

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 5.8.3 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 5.6 and 5.7 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 (such as HTTP and TLS, or CoAP and OSCORE [RFC8613]) 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 ciphersuite
   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 ciphersuites should be used.

   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 [RFC5077] 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, the use of session resumption is NOT RECOMMENDED for
   resource servers.

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

7.2. Multiple Access Tokens

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. Developers SHOULD avoid using multiple access tokens for a client.

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.

   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)", draft-ietf-ace-oauth-authz-35
              (work in progress), June 2020.

   [I-D.ietf-ace-oauth-params]
              Seitz, L., "Additional OAuth Parameters for Authorization
              in Constrained Environments (ACE)", draft-ietf-ace-oauth-
              params-13 (work in progress), April 2020.

   [I-D.ietf-cbor-7049bis]
              Bormann, C. and P. Hoffman, "Concise Binary Object
              Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work
              in progress), September 2020.

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

   [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>.

   [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>.

11.2. Informative References

   [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig,
              "Transport Layer Security (TLS) Session Resumption without
              Server-Side State", RFC 5077, DOI 10.17487/RFC5077,
              January 2008, <https://www.rfc-editor.org/info/rfc5077>.

   [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>.

   [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig,
              "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392,
              May 2018, <https://www.rfc-editor.org/info/rfc8392>.

   [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data
              Definition Language (CDDL): A Notational Convention to
              Express Concise Binary Object Representation (CBOR) and
              JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610,
              June 2019, <https://www.rfc-editor.org/info/rfc8610>.

   [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
              "Object Security for Constrained RESTful Environments
              (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019,
              <https://www.rfc-editor.org/info/rfc8613>.

Authors' Addresses

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

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

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

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

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

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

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Ludwig Seitz
Combitech
Djaeknegatan 31
Malmoe 211 35
Sweden

Email: ludwig.seitz@combitech.se

        Authentication and Authorization for Constrained Environments (ACE)
                  using the OAuth 2.0 Framework (ACE-OAuth)
                       draft-ietf-ace-oauth-authz-36

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.

Copyright Notice

Table of Contents

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 web
   applications on constrained nodes, this specification RECOMMENDS the
   use of the Constrained Application Protocol (CoAP) [RFC7252] as
   replacement for HTTP.

   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.

More detailed, interoperable specifications can be found in separate profile specifications. 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. Some devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of low end 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).

We use the term "Access Information" 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).

We use the term "Authorization Information" 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) [RFC7049], for encodings where JSON [RFC8259] is not
   sufficiently compact. CBOR is a binary encoding designed for small
   code and message size, which may be used for encoding of self
   contained tokens, and also for encoding payloads transferred in
   protocol messages.

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:

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 if it needs to request
   additional information regarding a received access token. The RS
   makes a POST request to the introspection endpoint on the AS and
   receives information about the access token in the response. (See
   "Introspection" below.)

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.

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 becomes invalid or 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 (PoP) 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 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 encrypted and
    included in the token. The PoP key is also encrypted for the
    token recipient and sent to the recipient together with the
    token.

Asymmetric PoP key:
    An asymmetric key pair is generated on the token's recipient
    and the public key is sent to the AS (if it does not already
    have knowledge of the recipient'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 requesting party. The
    consumer of the token can identify the public key from the
    information in the token, which allows the recipient of the
    token 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], an equivalent format using CBOR
   encoding (CWT) has been defined.

Introspection:
   Introspection is a method for a resource server 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].

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.

This framework RECOMMENDS the use of CoAP as replacement for HTTP 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.

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 types works
best depends on the usage scenario and [RFC7744] describes many
different IoT use cases but there are two preferred grant types,
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.

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 broadcasted
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].

```
+--------+                        +--------------+
|        |---(A)-- Token Request ------->|              |
|        |                        | Authorization |
|        |<--(B)-- Access Token ---------|    Server    |
|        |       + Access Information     |              |
|        |       + Refresh Token (optional) +--------------+
|        |                                 ^ |
|        |            Introspection Request (D)| |
| Client |                (optional)          | |
|        |                      Response      | |(E)
|        |                      (optional)    | v
|        |                                 +--------------+
|        |---(C)-- Token + Request ----->|              |
|        |                        |   Resource   |
|        |<--(F)-- Protected Resource ---|    Server    |
|        |                        |              |
+--------+                        +--------------+
```
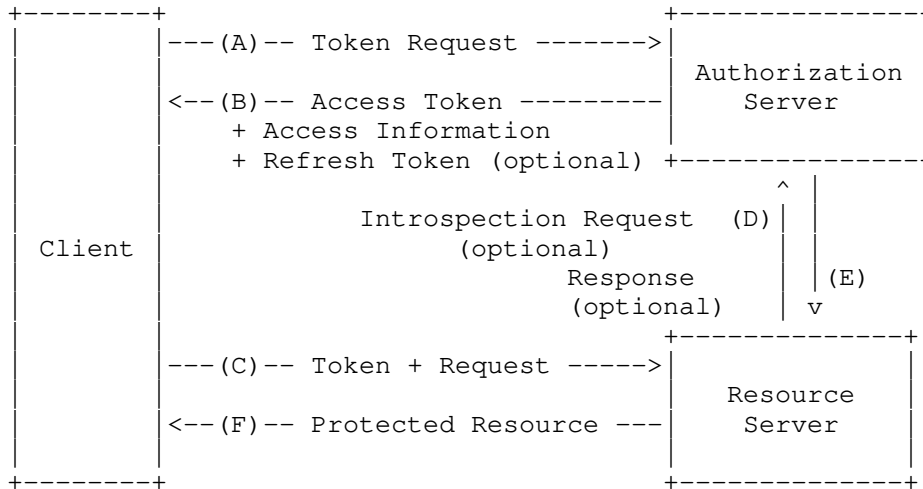
                    Figure 1: Basic Protocol Flow.

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
   (e.g., symmetric/asymmetric cryptography or a reference to a
   specific credential).

Access Token Response (B):
   If the AS successfully processes the request from the client, it
   returns an access token and optionally a refresh token (note that
   only certain grant types support refresh tokens). It 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 profiles 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, QUIC, MQTT, Bluetooth Low Energy, 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 may
      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.

5. Framework

   The following sections detail the profiling and extensions of OAuth
   2.0 for constrained environments, which constitutes the ACE
   framework.

   Credential Provisioning
      For IoT, it cannot be assumed that the client and RS are part of a
      common key infrastructure, so the AS provisions credentials or
      associated information to allow mutual authentication between
      client and RS. The resulting security association between client
      and 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 that provides
the features required 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 of the following alternative 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.

Profiles that use CBOR encoding of protocol message parameters at the
outermost encoding layer MUST use the media 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 [RFC7049] 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 it's owner if this AS is authorized for this RS. C may also use
a mechanism that addresses both problems at once.

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:

   o The request has been received on an unprotected channel.

   o The RS has no valid access token for the sender of the request
      regarding the requested action on that resource.

   o 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
   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:

   o A "audience" element containing a suggested audience that the
      client should request at the AS.

   o 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.

   o A "cnonce" element containing a client-nonce. See Section 5.3.1.

   o 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 [RFC7049] 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 (i.e., RS) uses an
internal clock that is only loosely 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.

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)
      7254656d7043                      # "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:

   o 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.

   o 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.

   o 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.

   o 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.erdtman-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, however, 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, JSON or CBOR payloads may be supported. If JSON
payloads are used, the semantics of Section 4 of the OAuth 2.0
specification MUST be followed (with additions as described below).
If CBOR payload is supported, the semantics described below 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 is '/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:

o 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.

o The "audience" parameter from [RFC8693] is OPTIONAL to request an
   access token bound to a specific audience.

o 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

o 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, specifically note that a binary
encoded scope does not necessarily use the space character '0x20'
to delimit scope-tokens.

o 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.

o 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 CBOR is used then these parameters MUST be provided as a CBOR map.

When HTTP is used as a transport then the client makes a request to
the token endpoint by sending the parameters using the "application/
x-www-form-urlencoded" format with a character encoding of UTF-8 in
the HTTP request entity-body, as defined in section 3.2 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'
  }
}W
```

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 CBOR payloads, the content MUST be encoded as a CBOR map,
containing parameters as specified in Section 5.1 of [RFC6749], 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 CoAP-based interactions with the AS are
   generally equivalent to the ones for HTTP-based interactions as
   defined in Section 5.2 of [RFC6749], with the following exceptions:

   o When using CBOR the raw payload before being processed by the
      communication security protocol MUST be encoded as a CBOR map.

   o 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].

   o The Content-Format (for CoAP-based interactions) or media type
      (for HTTP-based interactions) "application/ace+cbor" MUST be used
      for the error response.

   o The parameters "error", "error_description" and "error_uri" MUST
      be abbreviated using the codes specified in Figure 12, when a CBOR
      encoding is used.

   o 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 |
          |────────────────────────┼────────────|
          | invalid_request         |     1       |
          | invalid_client          |     2       |
          | invalid_grant           |     3       |
          | unauthorized_client     |     4       |
          | unsupported_grant_type  |     5       |
          | invalid_scope           |     6       |
          | unsupported_pop_key     |     7       |
          | incompatible_ace_profiles |   8       |
          \────────────────────────┴────────────/
```

             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:

   o 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" defined in
      Figure 10.

   o 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" defined 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     |        [RFC6749]       |
| authorization_code |     1     |        [RFC6749]       |
| client_credentials |     2     |        [RFC6749]       |
| refresh_token     |     3     |        [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.

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 (for JSON based interactions) 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 (Base64 encoded binary) for
JSON-based interactions. 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         |
      |-----------------+---------+--------------------|
      | access_token    | 1       | byte string        |
      | expires_in      | 2       | unsigned integer   |
      | audience        | 5       | text string        |
      | scope           | 9       | text or byte string|
      | client_id       | 24      | text string        |
      | client_secret   | 25      | byte string        |
      | response_type   | 26      | text string        |
      | redirect_uri    | 27      | text string        |
      | state           | 28      | text string        |
      | code            | 29      | byte string        |
      | error           | 30      | integer            |
      | error_description | 31    | text string        |
      | error_uri       | 32      | text string        |
      | grant_type      | 33      | unsigned integer   |
      | token_type      | 34      | integer            |
      | username        | 35      | text string        |
      | password        | 36      | text string        |
      | refresh_token   | 37      | byte string        |
      | ace_profile     | 38      | integer            |
      | cnonce          | 39      | byte string        |
      \-----------------+---------+--------------------/
```

         Figure 12: CBOR mappings used in token requests and responses

5.9. The Introspection Endpoint

   Token introspection [RFC7662] can be OPTIONALLY provided by the AS,
   and is then used by the RS and potentially the client 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 is '/introspect',
however implementations are not required to use this name and can
define their own instead.

The figures of this section uses CBOR diagnostic notation without the
integer abbreviations for the parameters or their values for better
readability.

Note that supporting introspection is OPTIONAL for implementations of
this framework.

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 CBOR 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", while for HTTP-based interactions the
equivalent media type "application/ace+cbor" MUST be 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 including with the same required and optional parameters as in
   Section 2.2 of [RFC7662] with the following addition:

   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.

   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.

   exi 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:

   o If content is sent and CBOR is used the payload MUST be encoded as
      a CBOR map and the Content-Format "application/ace+cbor" MUST be
      used.

   o 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 5.2 in [RFC6749].

   o 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.

   o The parameters "error", "error_description" and "error_uri" MUST
      be abbreviated using the codes specified in Figure 12.

   o 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            |
|------------------+----------+-----------------------|
| iss              | 1        | text string           |
| sub              | 2        | text string           |
| aud              | 3        | text string           |
| exp              | 4        | integer or            |
|                  |          |    floating-point number |
| nbf              | 5        | integer or            |
|                  |          |    floating-point number |
| iat              | 6        | integer or            |
|                  |          |    floating-point number |
| cti              | 7        | byte string           |
| scope            | 9        | text or byte string   |
| active           | 10       | True or False         |
| token            | 11       | byte string           |
| client_id        | 24       | text string           |
| error            | 30       | integer               |
| error_description| 31       | text string           |
| error_uri        | 32       | text string           |
| token_type_hint  | 33       | text string           |
| token_type       | 34       | integer               |
| username         | 35       | text string           |
| ace_profile      | 38       | integer               |
| cnonce           | 39       | byte string           |
| exi              | 40       | unsigned integer      |
\------------------+----------+-----------------------/
```

Figure 16: CBOR Mappings to Token Introspection Parameters.

5.10. The Access Token

This framework RECOMMENDS the use of CBOR web token (CWT) as
specified in [RFC8392].

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 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 2.01
(Created). Section 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.

This specification RECOMMENDS that an RS stores only one token per
proof-of-possession key, meaning that an additional token linked to
the same key will overwrite any existing token at the RS. The reason
is that this greatly simplifies (constrained) implementations, with

respect to required storage and resolving a request to the applicable token.

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 wrapper 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 a minimum, 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:

o If token or claim verification fails, 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).

o 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 must identify an AS that has the authority to issue access tokens for the receiving RS. 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 (exi)
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 and on it's children (if any).

The POST method SHOULD NOT be allowed on children of 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.

o 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.

o 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).

   o 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 "exi"
      ("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. For CBOR-based interaction this parameter is encoded as
      unsigned integer, while JSON-based interactions encode this as
      JSON number.

   o Processing this claim requires that the RS does the following:

      * For each token the RS receives, that contains an "exi" 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 "exi"
         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 "exi"
         claim is used:

         + 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 'exi' claim, issued
            by this AS for the RS.

         + The RS MUST store the highest sequence number of an expired
            token containing the "exi" claim that it has seen, and treat
            tokens with lower sequence numbers as expired.

   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:

o 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.

o 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 it's 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. Profiles MUST specify how communication security according to the requirements in Section 5 is provided.

   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).

Developers MUST ensure that the ephemeral credentials (i.e., the
private key or the session key) are not leaked to third parties. An
adversary in possession of the ephemeral 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 can often easily get physical access to the devices.
This risk can also be mitigated to some extent by making sure that
keys are refreshed more frequently.

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.

A compromised RS may use the hints for attempting to trick a client
into contacting an AS that is not supposed to be in charge of that
RS. Therefore, C must not communicate with an AS if it cannot
determine that this AS has the authority to issue access tokens for
this RS. Otherwise, a compromised RS may use this to perform a

denial of service attack against a specific AS, by redirecting a
large number of client requests to that AS.

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 on 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 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 "exi" claim, based on the
first time the RS sees the token was defined to provide a more
predictable alternative. The "exi" 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
   loose 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 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 were different profiles of this framework are
combined. For example, an MQTT-TLS profile is used between the
client and the RS in combination with a CoAP-DTLS profile for
interactions between the client and the AS. The security of a
profile MUST NOT depend on the assumption that the profile is used
for all the different types of interactions in this framework.

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 this framework
   RECOMMENDS 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 this framework RECOMMENDS 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:

   rt="ace.ai". This resource type describes an ACE-OAuth authz-info
   endpoint resource.

   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].

   o Error name: "unsupported_pop_key"
   o Error usage location: token error response
   o Related protocol extension: [this document]
   o Change Controller: IESG
   o Specification document(s): Section 5.8.3 of [this document]

   o Error name: "incompatible_ace_profiles"
   o Error usage location: token error response
   o Related protocol extension: [this document]
   o Change Controller: IESG
   o 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.

   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].

   o Type name: "PoP"
   o Additional Token Endpoint Response Parameters: "cnf", "rs_cnf" see
      section 3.3 of [I-D.ietf-ace-oauth-params].
   o HTTP Authentication Scheme(s): N/A
   o Change Controller: IETF
   o 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

   o Name: "Bearer"
   o Value: 1
   o Reference: [this document]
   o Original Specification: [RFC6749]

   o Name: "PoP"
   o Value: 2
   o Reference: [this document]
   o 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]:

   o Name: "ace_profile"
   o Parameter Usage Location: token response
   o Change Controller: IESG
   o 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.

   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].

   o Name: "ace_profile"
   o Description: The ACE profile used between client and RS.
   o Change Controller: IESG
   o Reference: Section 5.9.2 of [this document]

   o Name: "cnonce"
   o 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.
   o Change Controller: IESG
   o Reference: Section 5.9.2 of [this document]

   o Name: "exi"
   o Description: "Expires in". Lifetime of the token in seconds from
      the time the RS first sees it. Used to implement a weaker from of
      token expiration for devices that cannot synchronize their
      internal clocks.
   o Change Controller: IESG
   o 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.

   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]:

   o Claim Name: "ace_profile"
   o Claim Description: The ACE profile a token is supposed to be used
      with.
   o Change Controller: IESG
   o Reference: Section 5.10 of [this document]

   o Claim Name: "cnonce"
   o 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.
   o Change Controller: IESG
   o Reference: Section 5.10 of [this document]

   o Claim Name: "exi"
   o Claim Description: "Expires in". Lifetime of the token in seconds
      from the time the RS first sees it. Used to implement a weaker

      from of token expiration for devices that cannot synchronize their
      internal clocks.
   o Change Controller: IESG
   o 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].

   o Claim Name: "ace_profile"
   o Claim Description: The ACE profile a token is supposed to be used
      with.
   o JWT Claim Name: ace_profile
   o Claim Key: TBD (suggested: 38)
   o Claim Value Type(s): integer
   o Change Controller: IESG
   o Specification Document(s): Section 5.10 of [this document]

   o Claim Name: "cnonce"
   o Claim Description: The client-nonce sent to the AS by the RS via
      the client.
   o JWT Claim Name: cnonce
   o Claim Key: TBD (suggested: 39)
   o Claim Value Type(s): byte string
   o Change Controller: IESG
   o Specification Document(s): Section 5.10 of [this document]

   o Claim Name: "exi"
   o Claim Description: The expiration time of a token measured from
      when it was received at the RS in seconds.
   o JWT Claim Name: exi
   o Claim Key: TBD (suggested: 40)
   o Claim Value Type(s): integer
   o Change Controller: IESG
   o Specification Document(s): Section 5.10.3 of [this document]

   o Claim Name: "scope"
   o Claim Description: The scope of an access token as defined in
      [RFC6749].
   o JWT Claim Name: scope
   o Claim Key: TBD (suggested: 9)
   o Claim Value Type(s): byte string or text string
   o Change Controller: IESG
   o 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 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: IESG

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:

o 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.
o 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.
o 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.
o 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 large parts of the security considerations where 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).

    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.

    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)", draft-ietf-ace-oauth-
               params-13 (work in progress), April 2020.

    [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>.

   [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>.

   [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>.

   [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
              Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
              October 2013, <https://www.rfc-editor.org/info/rfc7049>.

   [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>.

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.erdtman-ace-rpcc]
              Seitz, L. and S. Erdtman, "Raw-Public-Key and Pre-Shared-
              Key as OAuth client credentials", draft-erdtman-ace-
              rpcc-02 (work in progress), October 2017.

   [I-D.ietf-quic-transport]
              Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed
              and Secure Transport", draft-ietf-quic-transport-32 (work
              in progress), October 2020.

   [I-D.ietf-tls-dtls13]
              Rescorla, E., Tschofenig, H., and N. Modadugu, "The
              Datagram Transport Layer Security (DTLS) Protocol Version
              1.3", draft-ietf-tls-dtls13-39 (work in progress),
              November 2020.

   [Margi10impact]
              Margi, C., de Oliveira, B., de Sousa, G., Simplicio Jr,
              M., Barreto, P., Carvalho, T., 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>.

   [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)
      [Margi10impact]. 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:

This framework RECOMMENDS the use of CBOR [RFC7049] as data
format. Where CBOR data needs to be protected, the use of COSE
[RFC8152] is RECOMMENDED. Furthermore, where self-contained
tokens are needed, this framework RECOMMENDS the 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:

This framework RECOMMENDS the 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:

This framework RECOMMENDS the use of the client credentials grant
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:

   This framework RECOMMENDS the use of access token introspection 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
   this framework RECOMMENDS the use of 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.

   o 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
   o Optionally specify new grant types. Section 5.4
   o Optionally define the use of client certificates as client
      credential type. Section 5.5

      o Specify the communication protocol the client and RS the must use
         (e.g., CoAP). Section 5 and Section 5.8.4.3
      o 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
      o Specify how the client and the RS mutually authenticate.
         Section 4
      o 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
      o Specify a unique ace_profile identifier. Section 5.8.4.3
      o If introspection is supported: Specify the communication and
         security protocol for introspection. Section 5.9
      o 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
      o Specify how/if the authz-info endpoint is protected, including how
         error responses are protected. Section 5.10.1
      o 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.

      o The identifier of the client or RS.
      o The profiles that the client or RS supports.
      o The scopes that the RS supports.
      o The audiences that the RS identifies with.
      o The key types (e.g., pre-shared symmetric key, raw public key, key
         length, other key parameters) that the client or RS supports.
      o The types of access tokens the RS supports (e.g., CWT).
      o If the RS supports CWTs, the COSE parameters for the crypto
         wrapper (e.g., algorithm, key-wrap algorithm, key-length) that the
         RS supports.
      o The expiration time for access tokens issued to this RS (unless
         the RS accepts a default time chosen by the AS).
      o The symmetric key shared between client and AS (if any).
      o The symmetric key shared between RS and AS (if any).
      o The raw public key of the client or RS (if any).
      o Whether the RS has synchronized time (and thus is able to use the
         'exp' claim) or not.

Appendix E. 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.

E.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 the 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.

```
             Authorization
   Client      Server
      |          |
      | <=======>|  DTLS Connection Establishment
      |          |     and mutual authentication
      |          |
A:    +--------->|  Header: POST (Code=0.02)
      |   POST   |  Uri-Path:"token"
      |          |  Content-Format: application/ace+cbor
      |          |  Payload: <Request-Payload>
      |          |
B:    |<--------+  Header: 2.05 Content
      |   2.05   |  Content-Format: application/ace+cbor
      |          |  Payload: <Response-Payload>
      |          |
```
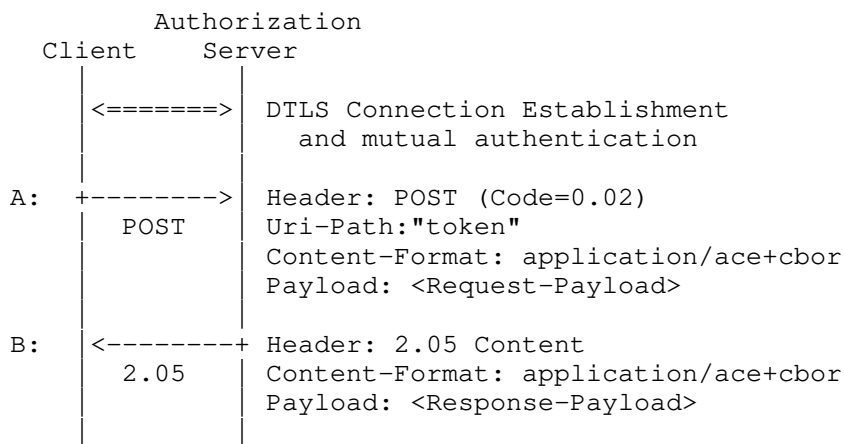
        Figure 17: Token Request and Response Using Client Credentials.

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'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
         "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'MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
         "y"   : b64'4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
       }
     }
   }
```

                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'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
        "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.

               Resource
     Client   Server
        | |
   C: +--------->| Header: POST (Code=0.02)
        | POST | Uri-Path:"authz-info"
        | | Payload: 0INDoQEKoQVN ...
        | |
        |<--------+ Header: 2.04 Changed
        | 2.04 |
        | |
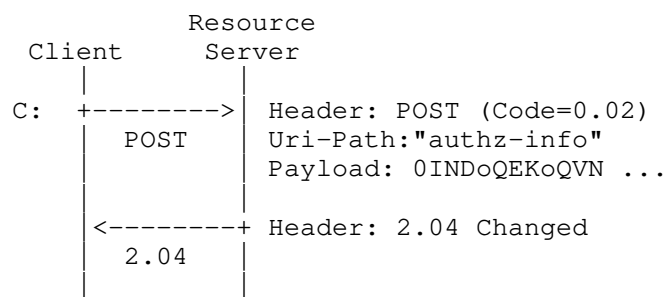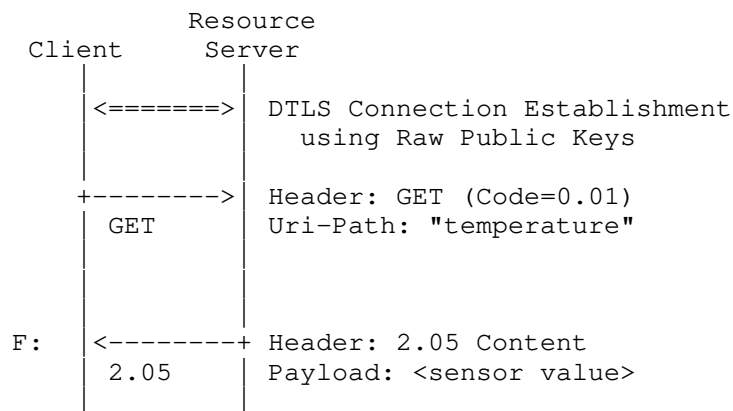
               Figure 20: Access Token provisioning to RS
      The client and the RS runs the DTLS handshake using the raw public
      keys established in step B and C.
      The client sends a CoAP GET request to /temperature on RS over
      DTLS. The RS verifies that the request is authorized, based on
      previously 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.

```
               Resource
     Client     Server
       |          |
       |<=======>|  DTLS Connection Establishment
       |          |     using Raw Public Keys
       |          |
       +-------->|  Header: GET (Code=0.01)
       |  GET     |  Uri-Path: "temperature"
       |          |
       |          |
       |          |
   F:  |<--------+  Header: 2.05 Content
       |  2.05    |  Payload: <sensor value>
       |          |
```

Figure 21: Resource Request and Response protected by DTLS.

E.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 used to authorize the key for his car at
the the car manufacturers 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 send 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 the 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 playload 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.

```
               Authorization
    Client        Server
       |           |
       |<=======>| DTLS Connection Establishment
       |           |     and mutual authentication
       |           |
 A:  +-------->| Header: POST (Code=0.02)
       |    POST    | Uri-Path:"token"
       |           | Content-Format: application/ace+cbor
       |           | Payload: <Request-Payload>
       |           |
 B:  |<--------+ Header: 2.05 Content
       |           | Content-Format: application/ace+cbor
       |    2.05    | Payload: <Response-Payload>
       |           |
```

        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'VGVzdCB0b2tlbg==',
     "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, rpk 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)
       |  POST  |  Uri-Path:"authz-info"
       |        |  Payload: b64'VGVzdCB0b2tlbg=='
       |        |
       |        |           Authorization
       |        |              Server
       |        |                |
       |    D: +--------->|  Header: POST (Code=0.02)
       |        |  POST    |  Uri-Path: "introspect"
       |        |          |  Content-Format: "application/ace+cbor"
       |        |          |  Payload: <Request-Payload>
       |        |          |
       |    E: |<---------+  Header: 2.05 Content
       |        |  2.05    |  Content-Format: "application/ace+cbor"
       |        |          |  Payload: <Response-Payload>
       |        |          |
       |        |
     |<--------+  Header: 2.01 Created
       |  2.01  |
       |        |
```

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'VGVzdCB0b2tlbg==',
  "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.

```
              Resource
    Client     Server
       |          |
       |<=======>|  DTLS Connection Establishment
       |          |      using Pre Shared Key
       |          |
       +-------->|  Header: PUT (Code=0.03)
       |  PUT     |  Uri-Path: "state"
       |          |  Payload: <new state for the lock>
       |          |
  F:   |<--------+  Header: 2.04 Changed
       |  2.04    |  Payload: <new state for the lock>
       |          |
```

        Figure 26: Resource request and response protected by OSCORE

Appendix F. Document Updates

   RFC EDITOR: PLEASE REMOVE THIS SECTION.

F.1. Version -21 to 22

     o Provided section numbers in references to OAuth RFC.
     o Updated IANA mapping registries to only use "Private Use" and
        "Expert Review".
     o Made error messages optional for RS at token submission since it
        may not be able to send them depending on the profile.
     o Corrected errors in examples.

F.2. Version -20 to 21

     o Added text about expiration of RS keys.

F.3. Version -19 to 20

     o Replaced "req_aud" with "audience" from the OAuth token exchange
        draft.
     o Updated examples to remove unnecessary elements.

F.4. Version -18 to -19

     o Added definition of "Authorization Information".
     o Explicitly state that ACE allows encoding refresh tokens in binary
        format in addition to strings.
     o Renamed "AS Information" to "AS Request Creation Hints" and added
        the possibility to specify req_aud and scope as hints.
     o Added the "kid" parameter to AS Request Creation Hints.
     o Added security considerations about the integrity protection of
        tokens with multi-RS audiences.
     o Renamed IANA registries mapping OAuth parameters to reflect the
        mapped registry.
     o Added JWT claim names to CWT claim registrations.
     o Added expert review instructions.
     o Updated references to TLS from 1.2 to 1.3.

F.5. Version -17 to -18

     o Added OSCORE options in examples involving OSCORE.
     o Removed requirement for the client to send application/cwt, since
        the client has no way to know.
     o Clarified verification of tokens by the RS.
     o Added exi claim CWT registration.

F.6. Version -16 to -17

     o Added references to (D)TLS 1.3.
     o Added requirement that responses are bound to requests.

      o Specify that grant_type is OPTIONAL in C2AS requests (as opposed
         to REQUIRED in OAuth).
      o Replaced examples with hypothetical COSE profile with OSCORE.
      o Added requirement for content type application/ace+cbor in error
         responses for token and introspection requests and responses.
      o Reworked abbreviation space for claims, request and response
         parameters.
      o Added text that the RS may indicate that it is busy at the authz-
         info resource.
      o Added section that specifies how the RS verifies an access token.
      o Added section on the protection of the authz-info endpoint.
      o Removed the expiration mechanism based on sequence numbers.
      o Added reference to RFC7662 security considerations.
      o Added considerations on minimal security requirements for
         communication.
      o Added security considerations on unprotected information sent to
         authz-info and in the error responses.

F.7. Version -15 to -16

      o Added text the RS using RFC6750 error codes.
      o Defined an error code for incompatible token request parameters.
      o Removed references to the actors draft.
      o Fixed errors in examples.

F.8. Version -14 to -15

      o Added text about refresh tokens.
      o Added text about protection of credentials.
      o Rephrased introspection so that other entities than RS can do it.
      o Editorial improvements.

F.9. Version -13 to -14

      o Split out the 'aud', 'cnf' and 'rs_cnf' parameters to
         [I-D.ietf-ace-oauth-params]
      o Introduced the "application/ace+cbor" Content-Type.
      o Added claim registrations from 'profile' and 'rs_cnf'.
      o Added note on schema part of AS Information Section 5.3
      o Realigned the parameter abbreviations to push rarely used ones to
         the 2-byte encoding size of CBOR integers.

F.10. Version -12 to -13

      o Changed "Resource Information" to "Access Information" to avoid
         confusion.
      o Clarified section about AS discovery.
      o Editorial changes

F.11. Version -11 to -12

   o Moved the Request error handling to a section of its own.
   o Require the use of the abbreviation for profile identifiers.
   o Added rs_cnf parameter in the introspection response, to inform
      RS' with several RPKs on which key to use.
   o Allowed use of rs_cnf as claim in the access token in order to
      inform an RS with several RPKs on which key to use.
   o Clarified that profiles must specify if/how error responses are
      protected.
   o Fixed label number range to align with COSE/CWT.
   o Clarified the requirements language in order to allow profiles to
      specify other payload formats than CBOR if they do not use CoAP.

F.12. Version -10 to -11

   o Fixed some CBOR data type errors.
   o Updated boilerplate text

F.13. Version -09 to -10

   o Removed CBOR major type numbers.
   o Removed the client token design.
   o Rephrased to clarify that other protocols than CoAP can be used.
   o Clarifications regarding the use of HTTP

F.14. Version -08 to -09

   o Allowed scope to be byte strings.
   o Defined default names for endpoints.
   o Refactored the IANA section for briefness and consistency.
   o Refactored tables that define IANA registry contents for
      consistency.
   o Created IANA registry for CBOR mappings of error codes, grant
      types and Authorization Server Information.
   o Added references to other document sections defining IANA entries
      in the IANA section.

F.15. Version -07 to -08

   o Moved AS discovery from the DTLS profile to the framework, see
      Section 5.1.
   o Made the use of CBOR mandatory. If you use JSON you can use
      vanilla OAuth.
   o Made it mandatory for profiles to specify C-AS security and RS-AS
      security (the latter only if introspection is supported).
   o Made the use of CBOR abbreviations mandatory.

   o Added text to clarify the use of token references as an
      alternative to CWTs.
   o Added text to clarify that introspection must not be delayed, in
      case the RS has to return a client token.
   o Added security considerations about leakage through unprotected AS
      discovery information, combining profiles and leakage through
      error responses.
   o Added privacy considerations about leakage through unprotected AS
      discovery.
   o Added text that clarifies that introspection is optional.
   o Made profile parameter optional since it can be implicit.
   o Clarified that CoAP is not mandatory and other protocols can be
      used.
   o Clarified the design justification for specific features of the
      framework in appendix A.
   o Clarified appendix E.2.
   o Removed specification of the "cnf" claim for CBOR/COSE, and
      replaced with references to [RFC8747]

F.16. Version -06 to -07

   o Various clarifications added.
   o Fixed erroneous author email.

F.17. Version -05 to -06

   o Moved sections that define the ACE framework into a subsection of
      the framework Section 5.
   o Split section on client credentials and grant into two separate
      sections, Section 5.4, and Section 5.5.
   o Added Section 5.6 on AS authentication.
   o Added Section 5.7 on the Authorization endpoint.

F.18. Version -04 to -05

   o Added RFC 2119 language to the specification of the required
      behavior of profile specifications.
   o Added Section 5.5 on the relation to the OAuth2 grant types.
   o Added CBOR abbreviations for error and the error codes defined in
      OAuth2.
   o Added clarification about token expiration and long-running
      requests in Section 5.10.3
   o Added security considerations about tokens with symmetric PoP keys
      valid for more than one RS.
   o Added privacy considerations section.
   o Added IANA registry mapping the confirmation types from RFC 7800
      to equivalent COSE types.

   o Added appendix D, describing assumptions about what the AS knows
      about the client and the RS.

F.19. Version -03 to -04

   o Added a description of the terms "framework" and "profiles" as
      used in this document.
   o Clarified protection of access tokens in section 3.1.
   o Clarified uses of the "cnf" parameter in section 6.4.5.
   o Clarified intended use of Client Token in section 7.4.

F.20. Version -02 to -03

   o Removed references to draft-ietf-oauth-pop-key-distribution since
      the status of this draft is unclear.
   o Copied and adapted security considerations from draft-ietf-oauth-
      pop-key-distribution.
   o Renamed "client information" to "RS information" since it is
      information about the RS.
   o Clarified the requirements on profiles of this framework.
   o Clarified the token endpoint protocol and removed negotiation of
      "profile" and "alg" (section 6).
   o Renumbered the abbreviations for claims and parameters to get a
      consistent numbering across different endpoints.
   o Clarified the introspection endpoint.
   o Renamed token, introspection and authz-info to "endpoint" instead
      of "resource" to mirror the OAuth 2.0 terminology.
   o Updated the examples in the appendices.

F.21. Version -01 to -02

   o Restructured to remove communication security parts. These shall
      now be defined in profiles.
   o Restructured section 5 to create new sections on the OAuth
      endpoints token, introspection and authz-info.
   o Pulled in material from draft-ietf-oauth-pop-key-distribution in
      order to define proof-of-possession key distribution.
   o Introduced the "cnf" parameter as defined in RFC7800 to reference
      or transport keys used for proof of possession.
   o Introduced the "client-token" to transport client information from
      the AS to the client via the RS in conjunction with introspection.
   o Expanded the IANA section to define parameters for token request,
      introspection and CWT claims.
   o Moved deployment scenarios to the appendix as examples.

F.22. Version -00 to -01

   o Changed 5.1. from "Communication Security Protocol" to "Client
      Information".
   o Major rewrite of 5.1 to clarify the information exchanged between
      C and AS in the PoP access token request profile for IoT.

      * Allow the client to indicate preferences for the communication
         security protocol.
      * Defined the term "Client Information" for the additional
         information returned to the client in addition to the access
         token.
      * Require that the messages between AS and client are secured,
         either with (D)TLS or with COSE_Encrypted wrappers.
      * Removed dependency on OSCOAP and added generic text about
         object security instead.
      * Defined the "rpk" parameter in the client information to
         transmit the raw public key of the RS from AS to client.
      * (D)TLS MUST use the PoP key in the handshake (either as PSK or
         as client RPK with client authentication).
      * Defined the use of x5c, x5t and x5tS256 parameters when a
         client certificate is used for proof of possession.
      * Defined "tktn" parameter for signaling for how to transfer the
         access token.
   o Added 5.2. the CoAP Access-Token option for transferring access
      tokens in messages that do not have payload.
   o 5.3.2. Defined success and error responses from the RS when
      receiving an access token.
   o 5.6.:Added section giving guidance on how to handle token
      expiration in the absence of reliable time.
   o Appendix B Added list of roles and responsibilities for C, AS and
      RS.

Authors' Addresses

   Ludwig Seitz
   Combitech
   Djaeknegatan 31
   Malmoe 211 35
   Sweden

   Email: ludwig.seitz@combitech.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

Samuel Erdtman
Spotify AB
Birger Jarlsgatan 61, 4tr
Stockholm 113 56
Sweden

Email: erdtman@spotify.com

Hannes Tschofenig
Arm Ltd.
Absam 6067
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group   F. Palombini
Internet-Draft   Ericsson AB
Intended status: Standards Track   L. Seitz
Expires: April 30, 2021   Combitech
                                                          G. Selander
                                                          Ericsson AB
                                                        M. Gunnarsson
                                                                 RISE
                                                     October 27, 2020

OSCORE Profile of the Authentication and Authorization for Constrained
                         Environments Framework
                    draft-ietf-ace-oscore-profile-13

Abstract

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

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 April 30, 2021.

Copyright Notice

Table of Contents

1. Introduction

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

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

1.1. Terminology

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

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

   RESTful terminology follows HTTP [RFC7231].

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

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

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

2. Protocol Overview

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

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

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

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

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

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

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

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

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

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

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

```
        C                            RS                   AS
        |                            |                    |
        | ----- POST /token ---------------------------> |
        |                            |                    |
        | <------------------------- Access Token ------  |
        |                             + Access Information |
        | ---- POST /authz-info --->  |                    |
        |    (access_token, N1, ID1)  |                    |
        |                            |                    |
        | <- 2.01 Created (N2, ID2)-  |                    |
        |                            |                    |
     /Sec Context             /Sec Context                |
      derivation/                derivation/              |
        |                            |                    |
        | ---- OSCORE Request ----->  |                    |
        |                            |                    |
        |                        /proof-of-possession      |
        |                        Sec Context storage/       |
        |                            |                    |
        | <--- OSCORE Response -----  |                    |
        |                            |                    |
     /proof-of-possession            |                    |
     Sec Context storage/            |                    |
        |                            |                    |
        | ---- OSCORE Request ----->  |                    |
        |                            |                    |
        | <--- OSCORE Response -----  |                    |
        |            ...              |                    |
```

                     Figure 1: Protocol Overview

3. Client-AS Communication

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

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

   The client-to-AS request is specified in Section 5.6.1 of
   [I-D.ietf-ace-oauth-authz].

The client must send this POST request to the token endpoint over a
secure channel that guarantees authentication, message integrity and
confidentiality (see Section 5).

An example of such a request, with payload in CBOR diagnostic
notation without the tag and value abbreviations is reported in
Figure 2

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

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

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

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

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

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

3.2. AS-to-C: Access Token

   After verifying the POST request to the token endpoint and that the
   client is authorized to obtain an access token corresponding to its
   access token request, the AS responds as defined in section 5.6.2 of
   [I-D.ietf-ace-oauth-authz]. If the client request was invalid, or
   not authorized, the AS returns an error response as described in
   section 5.6.3 of [I-D.ietf-ace-oauth-authz].

   The AS can signal that the use of OSCORE is REQUIRED for a specific
   access token by including the "profile" parameter with the value
   "coap_oscore" in the access token response. This means that the
   client MUST use OSCORE towards all resource servers for which this
   access token is valid, and follow Section 4.3 to derive the security
   context to run OSCORE. Usually it is assumed that constrained
   devices will be pre-configured with the necessary profile, so that
   this kind of profile negotiation can be omitted.

   Moreover, the AS MUST send the following data:

   o a master secret

   o an identifier of the OSCORE Input Material

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

   o a context identifier

   o an AEAD algorithm

   o an HMAC-based key derivation function (HKDF) algorithm

o a salt

o the OSCORE version number

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

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

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

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

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

Figure 4 shows an example of an AS response, with payload in CBOR
diagnostic notation without the tag and value abbreviations. The
access token has been truncated for readability.

```
     Header: Created (Code=2.01)
     Content-Type: "application/ace+cbor"
     Payload:
     {
       "access_token" : h'8343a1010aa2044c53 ...
         (remainder of access token (CWT) omitted for brevity)',
       "profile" : "coap_oscore",
       "expires_in" : "3600",
       "cnf" : {
         "osc" : {
           "id" : h'01',
           "ms" : h'f9af838368e353e78888e1426bd94e6f'
         }
       }
     }
```

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

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

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

        Figure 5: Example CWT Claims Set with OSCORE parameters.

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

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

```
A5                                          # map(5)
   63                                       # text(3)
      617564                                # "aud"
   76                                       # text(22)
      74656D7053656E736F72496E4C6976696E67526F6F6D
                                            # "tempSensorInLivingRoom"
   63                                       # text(3)
      696174                                # "iat"
   6A                                       # text(10)
      31333630313839323234                  # "1360189224"
   63                                       # text(3)
      657870                                # "exp"
   6A                                       # text(10)
      31333630323839323234                  # "1360289224"
   65                                       # text(5)
      73636F7065                            # "scope"
   78 18                                    # text(24)
      74656D70657261747572655F67206669726D776172655F70
                                            # "temperature_g firmware_p"
   63                                       # text(3)
      636E66                                # "cnf"
   A1                                       # map(1)
      63                                    # text(3)
         6F7363                             # "osc"
      A2                                    # map(2)
         62                                 # text(2)
            6D73                            # "ms"
         50                                 # bytes(16)
            F9AF838368E353E78888E1426BD94E6F
                                            # "\xF9\xAF\x83\x83h\xE3S\xE7
                                              \x88\x88\xE1Bk\xD9No"
         62                                 # text(2)
            6964                            # "id"
         41                                 # bytes(1)
            01                              # "\x01"
```

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

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

Figure 7 shows an example of such an AS response, with payload in
CBOR diagnostic notation without the tag and value abbreviations.
The access token has been truncated for readability.

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

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

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

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

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

3.2.1. The OSCORE_Input_Material

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

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

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

Table 1: OSCORE_Input_Material Parameters

version: This parameter identifies the OSCORE Version number, which
   is an unsigned integer. For more information about this field,
   see section 5.4 of [RFC8613]. In JSON, the "version" value is an
   integer. In CBOR, the "version" type is int, and has label 0.

ms: This parameter identifies the OSCORE Master Secret value, which
   is a byte string. For more information about this field, see

   section 3.1 of [RFC8613]. In JSON, the "ms" value is a Base64
   encoded byte string. In CBOR, the "ms" type is bstr, and has
   label 1.

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

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

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

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

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

An example of JSON OSCORE_Input_Material is given in Figure 9.

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

Figure 9: Example JSON OSCORE_Input_Material

The CDDL grammar describing the CBOR OSCORE_Input_Material is:

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

4. Client-RS Communication

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

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

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

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

   The client generates its own Recipient ID, ID1, for the OSCORE
   Security Context that it is establishing with the RS. By generating
   its own Recipient ID, the client makes sure that it does not collide
   with any of its Recipient IDs.

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

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

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

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

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

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

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

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

4.1.1. The Nonce 1 Parameter

This parameter MUST be sent from the client to the RS, together with
the access token, if the ace profile used is coap_oscore. The
parameter is encoded as a byte string for CBOR-based interactions,
and as a string (Base64 encoded binary) for JSON-based interactions.
This parameter is registered in Section 9.2.

4.1.2. The ace_client_recipientid Parameter

This parameter MUST be sent from the client to the RS, together with
the access token, if the ace profile used is coap_oscore. The
parameter is encoded as a byte string for CBOR-based interactions,
and as a string (Base64 encoded binary) for JSON-based interactions.
This parameter is registered in Section 9.2.

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

The RS MUST follow the procedures defined in section 5.8.1 of
[I-D.ietf-ace-oauth-authz]: the RS must verify the validity of the
token. If the token is valid, the RS must respond to the POST
request with 2.01 (Created). If the token is valid but is associated

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

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

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

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

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

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

If the RS receives the token in a OSCORE protected message, it means that the client is requesting an update of access rights. The RS MUST discard any nonce and identifiers in the request, if any was sent. The RS MUST check that the "kid" of the "cnf" parameter of the

new access token matches the OSCORE Input Material of the context
used to protect the message. If that is the case, the RS MUST
discard the old token and associate the new token to the Security
Context identified by the "kid" value in the "cnf" parameter. The RS
MUST respond with a 2.01 (Created) response protected with the same
Security Context, with no payload. If any verification fails, the RS
MUST respond with a 4.01 (Unauthorized) error response.

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

4.2.1. The Nonce 2 Parameter

This parameter MUST be sent from the RS to the client if the ace
profile used is coap_oscore. The parameter is encoded as a byte
string for CBOR-based interactions, and as a string (Base64 encoded
binary) for JSON-based interactions. This parameter is registered in
Section 9.2

4.2.2. The ace_server_recipientid Parameter

This parameter MUST be sent from the RS to the client if the ace
profile used is coap_oscore. The parameter is encoded as a byte
string for CBOR-based interactions, and as a string (Base64 encoded
binary) for JSON-based interactions. This parameter is registered in
Section 9.2

4.3. OSCORE Setup

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

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

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

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

    Figure 12: Example of Master Salt construction using CBOR encoding

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

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

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

Master Salt = b64'EPmvg4No41PniIjhQmvZTm8IAYonj3+qtVoIJaiZHNcArAE='

    Figure 13: Example of Master Salt construction using Base64 encoding

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

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

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

After sending the 2.01 (Created) response, the RS MUST set the Master
Salt of the Security Context created to communicate with the client
to the concatenation of salt, N1, and N2, in the same way described
above. An example of Master Salt construction using CBOR encoding is
given in Figure 12 and using Base64 encoding is given in Figure 13.
The RS MUST set the Sender ID from the ace_client_recipientid
received in Section 4.1, and the Recipient ID from the
ace_server_recipientid sent in Section 4.2. The RS MUST set the
Master Secret from the parameter, received from the AS and forwarded
by the client in the access token in Section 4.1 after validation of
the token as specified in Section 4.2. The RS MUST set the AEAD
Algorithm, ID Context, HKDF, and OSCORE Version from the parameters
received from the AS and forwarded by the client in the access token
in Section 4.1 after validation of the token as specified in
Section 4.2, if present. In case an optional parameter is omitted,
the default value SHALL be used as described in sections 3.2 and 5.4
of [RFC8613]. After that, the RS MUST derive the complete Security
Context following section 3.2.1 of [RFC8613], and MUST associate this
Security Context with the authorization information from the access
token.

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

Note that the ID Context can be assigned by the AS, communicated and
set in both the RS and client after the exchange specified in this
profile is executed. Subsequently, client and RS can update their ID
Context by running a mechanism such as the one defined in
Appendix B.2 of [RFC8613] if they both support it and are configured
to do so. In that case, the ID Context in the OSCORE Security
Context will not match the "contextId" parameter of the corresponding

OSCORE_Input_Material. Running Appendix B.2 results in the keying
material in the Security Contexts of client and RS being updated;
this same result can also be achieved by the client reposting the
access token as described in Section 4.1, but without updating the ID
Context.

4.4. Access rights verification

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

5. Secure Communication with AS

As specified in the ACE framework (section 5.7 of
[I-D.ietf-ace-oauth-authz]), the requesting entity (RS and/or client)
and the AS communicates via the introspection or token endpoint. The
use of CoAP and OSCORE ([RFC8613]) for this communication is
RECOMMENDED in this profile, other protocols (such as HTTP and DTLS
or TLS) MAY be used instead.

If OSCORE is used, the requesting entity and the AS are expected to
have pre-established security contexts in place. How these security
contexts are established is out of scope for this profile.
Furthermore the requesting entity and the AS communicate through the
introspection endpoint as specified in section 5.7 of
[I-D.ietf-ace-oauth-authz] and through the token endpoint as
specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

6. Discarding the Security Context

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

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

o the Sequence Number space ends.

o the access token associated with the context expires.

   o the client receives a number of 4.01 Unauthorized responses to
      OSCORE requests using the same Security Context. The exact number
      needs to be specified by the application.

   o the client receives a new nonce in the 2.01 (Created) response
      (see Section 4.2) to a POST request to the authz-info endpoint,
      when re-posting a (non-expired) token associated to the existing
      context.

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

   o the Sequence Number space ends.

   o the access token associated with the context expires.

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

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

7. Security Considerations

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

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

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

authentication requires both that the client trusts the AS and that the OSCORE response from the RS pass verification.

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

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

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

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

8. Privacy Considerations

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

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

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

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

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

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

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

9. IANA Considerations

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

9.1. ACE Profile Registry

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

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

9.2. OAuth Parameters Registry

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

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

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

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

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

9.3. OAuth Parameters CBOR Mappings Registry

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

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

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

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

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

9.4. OSCORE Security Context Parameters Registry

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

   The columns of the registry are:

   name The JSON name requested (e.g., "ms"). Because a core goal of
      this specification is for the resulting representations to be
      compact, it is RECOMMENDED that the name be short. This name is
      case sensitive. Names may not match other registered names in a
      case-insensitive manner unless the Designated Experts determine
      that there is a compelling reason to allow an exception. The name
      is not used in the CBOR encoding.
   CBOR label The value to be used to identify this algorithm. Map key
      labels MUST be unique. The label can be a positive integer, a
      negative integer or a string. Integer values between -256 and 255
      and strings of length 1 are designated as Standards Track Document
      required. Integer values from -65536 to -257 and from 256 to
      65535 and strings of length 2 are designated as Specification
      Required. Integer values greater than 65535 and strings of length
      greater than 2 are designated as expert review. Integer values
      less than -65536 are marked as private use.
   CBOR Type This field contains the CBOR type for the field.
   registry This field denotes the registry that values may come from,
      if one exists.
   description This field contains a brief description for the field.
   specification This contains a pointer to the public specification
      for the field if one exists

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

9.5. CWT Confirmation Methods Registry

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

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

9.6. JWT Confirmation Methods Registry

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

   o Confirmation Method Value: "osc"
   o Confirmation Method Description: OSCORE_Input_Material carrying
      the parameters for using OSCORE per-message security with implicit
      key confirmation
   o Change Controller: IESG
   o Specification Document(s): Section 3.2.1 of [[this specification]]

9.7. Expert Review Instructions

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

   Expert reviewers should take into consideration the following points:

   o 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.
   o 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.

o Experts should take into account the expected usage of fields when
approving point assignment. The fact that there is a range for
standards track documents does not mean that a standards track
document cannot have points assigned outside of that range. The
length of the encoded value should be weighed against how many
code points of that length are left, the size of device it will be
used on, and the number of code points left that encode to that
size.

10. References

10.1. Normative References

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

   [I-D.ietf-ace-oauth-authz]
              Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
              H. Tschofenig, "Authentication and Authorization for
              Constrained Environments (ACE) using the OAuth 2.0
              Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-35
              (work in progress), June 2020.

   [I-D.ietf-ace-oauth-params]
              Seitz, L., "Additional OAuth Parameters for Authorization
              in Constrained Environments (ACE)", draft-ietf-ace-oauth-
              params-13 (work in progress), April 2020.

   [I-D.ietf-cbor-7049bis]
              Bormann, C. and P. Hoffman, "Concise Binary Object
              Representation (CBOR)", draft-ietf-cbor-7049bis-16 (work
              in progress), September 2020.

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

   [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
              Application Protocol (CoAP)", RFC 7252,
              DOI 10.17487/RFC7252, June 2014,
              <https://www.rfc-editor.org/info/rfc7252>.

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

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

   [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig,
              "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392,
              May 2018, <https://www.rfc-editor.org/info/rfc8392>.

   [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data
              Definition Language (CDDL): A Notational Convention to
              Express Concise Binary Object Representation (CBOR) and
              JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610,
              June 2019, <https://www.rfc-editor.org/info/rfc8610>.

   [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
              "Object Security for Constrained RESTful Environments
              (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019,
              <https://www.rfc-editor.org/info/rfc8613>.

10.2. Informative References

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

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

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

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

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

Appendix A. Profile Requirements

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

   o Optionally define new methods for the client to discover the
      necessary permissions and AS for accessing a resource, different
      from the one proposed in: Not specified
   o Optionally specify new grant types: Not specified
   o Optionally define the use of client certificates as client
      credential type: Not specified
   o Specify the communication protocol the client and RS the must use:
      CoAP
   o Specify the security protocol the client and RS must use to
      protect their communication: OSCORE
   o Specify how the client and the RS mutually authenticate:
      Implicitly by possession of a common OSCORE security context.
      Note that the mutual authentication is not completed before the
      client has verified an OSCORE response using this security
      context.
   o Specify the proof-of-possession protocol(s) and how to select one,
      if several are available. Also specify which key types (e.g.,
      symmetric/asymmetric) are supported by a specific proof-of-
      possession protocol: OSCORE algorithms; pre-established symmetric
      keys
   o Specify a unique ace_profile identifier: coap_oscore
   o If introspection is supported: Specify the communication and
      security protocol for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE)
   o Specify the communication and security protocol for interactions
      between client and AS: HTTP/CoAP (+ TLS/DTLS/OSCORE)
   o Specify how/if the authz-info endpoint is protected, including how
      error responses are protected: Not protected.
   o Optionally define other methods of token transport than the authz-
      info endpoint: Not defined

Acknowledgments

Authors' Addresses

   Francesca Palombini
   Ericsson AB

   Email: francesca.palombini@ericsson.com

   Ludwig Seitz
   Combitech
   Djaeknegatan 31
   Malmoe 211 35
   Sweden

   Email: ludwig.seitz@combitech.se

   Goeran Selander
   Ericsson AB

   Email: goran.selander@ericsson.com

   Martin Gunnarsson
   RISE
   Scheelevagen 17
   Lund 22370
   Sweden

   Email: martin.gunnarsson@ri.se

ACE Working Group   F. Palombini
Internet-Draft   Ericsson
Intended status: Standards Track   November 04, 2019
Expires: May 7, 2020

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

Abstract

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

Status of This Memo

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

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

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

   This Internet-Draft will expire on May 7, 2020.

Table of Contents

1. Introduction

   The publisher-subscriber setting allows for devices with limited
   reachability to communicate via a broker that enables store-and-
   forward messaging between the devices. The pub-sub scenario using
   the Constrained Application Protocol (CoAP) is specified in
   [I-D.ietf-core-coap-pubsub]. This document defines a way to
   authorize nodes in a CoAP pub-sub type of setting, using the ACE
   framework [I-D.ietf-ace-oauth-authz], and to provide the keys for
   protecting the communication between these nodes.

1.1. Terminology

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

   Readers are expected to be familiar with the terms and concepts
   described in [I-D.ietf-ace-oauth-authz], [I-D.ietf-ace-key-groupcomm]
   and [I-D.ietf-core-coap-pubsub]. In particular, analogously to
   [I-D.ietf-ace-oauth-authz], terminology for entities in the
   architecture such as Client (C), Resource Server (RS), and

Authorization Server (AS) is defined in OAuth 2.0 [RFC6749] and
[I-D.ietf-ace-actors], and terminology for entities such as the Key
Distribution Center (KDC) and Dispatcher in
[I-D.ietf-ace-key-groupcomm].

2. Application Profile Overview

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

The architecture of the scenario is shown in Figure 1.

```
          +----------------+  +----------------+
          |                |  |                |
          |  Authorization |  |  Authorization |
          |    Server 1    |  |    Server 2    |
          |                |  |                |
          +--------------+-+  +--------------+-+
                    ^                  ^   ^
                    |                  |   |
          +--------- (A) ----+         |   +----- (D) ------+
          |   +------------------- (B) --------+            |
          v   v                      v                      v
        +-----------+            +-----------+          +-----------+
        |   CoAP    | ----(C)--->|   CoAP    |          |   CoAP    |
        |  Client - | ----(E)--->|  Server - |          |  Client - |
        |           |            |           | <----(F)----          |
        | Publisher |            |  Broker   | -----(G)--->| Subscriber|
        +-----------+            +-----------+          +-----------+
```

Figure 1: Architecture CoAP pubsub with Authorization Servers

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

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

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

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

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

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

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

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

```
   +-----------+            +-----------+            +-----------+
   |   CoAP    |            |   CoAP    |            |   CoAP    |
   |  Client - |            |  Server - |            |  Client - |
   |           |            |           |            |           |
   | Publisher |            |  Broker   |            | Subscriber|
   +-----------+            +-----------+            +-----------+
        :    :                    :                        :
        :    '------ Security -------'                      :
        :           Association 1                           :
        '---------------------------- Security -------------'
                                     Association 2
```

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

3. coap_pubsub_app Application Profile

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

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

    Note that both publishers and subscribers use the same profile,
    called "coap_pubsub_app".

3.1. Retrieval of COSE Key for protection of content

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

```
          Client                          Broker          AS2
             │ [----- Resource Request ---->] │             │
             │                                │             │
             │ [<-- AS1, AS2 Information ---] │             │
             │                                │             │
             │ [------ Pub Key Format Negociation Request --->] │
             │                                │             │
             │ [<---- Pub Key Format Negociation Response ----] │
             │                                │             │
             │ -- Authorization + Key Distribution Request ---> │
             │                                │             │
             │ <-- Authorization + Key Distribution Response -- │
             │                                │             │
```

                 Figure 2: B: Access request - response

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

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

               Figure 3: AS1, AS2 Information example

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

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

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

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

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

         + the broker's topic as first element, and

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

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

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

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

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

* OPTIONALLY, if needed, the 'pub_keys_repos' parameter

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

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

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

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

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

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

o the following fields from the Joining Response (Section 4.1 of
  [I-D.ietf-ace-key-groupcomm]):

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

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

        + 'kty' with value 4 (symmetric)

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

+ 'Base IV' with value defined by the AS2

+ 'k' with value the symmetric key value

+ OPTIONALLY, 'kid' with an identifier for the key value

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

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

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

* 'profile' set to "coap_pubsub_app", as specified in Section 8.1

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

+ the broker's topic as first element, and

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

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

4. Publisher

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

```
          +---------------+   +---------------+
          |               |   |               |
          | Authorization |   | Authorization |
          |    Server 1   |   |    Server 2   |
          |               |   |               |
          +---------------+   +---------------+
                  ^                   ^
                  |                   |
        +---------(A)----+           |
        |  +------------------(B)--------+
        v  v
   +-----------+             +-----------+
   |    CoAP   | ----(C)---> |    CoAP   |
   |  Client - |             |  Server - |
   |           |             |           |
   | Publisher |             |   Broker  |
   +-----------+             +-----------+
```

                 Figure 4: Phase 1: Publisher side

   This is a combination of two independent phases:

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

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

   In detail:

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

   (C) corresponds to the exchange between Publisher and Broker, where
   the Publisher sends its access token to the Broker and establishes a
   secure connection with the Broker. Depending on the Information
   received in (A), this can be for example DTLS handshake, or other
   protocols. Depending on the application, there may not be the need
   for this set up phase: for example, if OSCORE is used directly.

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

   (B) corresponds to the retrieval of the keying material to protect
   the publication end-to-end with the subscribers (see Section 6.1),

and uses [I-D.ietf-ace-key-groupcomm]. The details are defined in
Section 3.1.

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

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

Figure 5: Authorization + Joining Request payload for a Publisher

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

Figure 6: Authorization + Joining Response payload for a Publisher

5. Subscriber

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

```
                    +---------------+
                    |               |
                    | Authorization |
                    |   Server 2    |
                    |               |
                    +---------------+
                           ^
                           |
                    +----- (D) ------+
                                     |
                                     v
                                +-----------+
                                |           |
                                |   CoAP    |
                                | Client -  |
                                |           |
                                | Subscriber|
                                |           |
                                +-----------+
```

                  Figure 7: Phase 2: Subscriber side

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

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

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

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

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

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

    Figure 8: Authorization + Joining Request payload for a Subscriber

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

        Figure 9: Authorization + Joining Response payload for a Subscriber

6. Pub-Sub Protected Communication

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

```
   +------------+              +------------+              +------------+
   |   CoAP     |              |   CoAP     |              |   CoAP     |
   |  Client -  |              |  Server -  |              |  Client -  |
   |            | ----(E)--->  |            |              |            |
   |  Publisher |              |   Broker   | <----(F)---- |  Subscriber|
   |            |              |            | -----(G)---> |            |
   +------------+              +------------+              +------------+
```

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

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

The flow graph is presented below.

```
        Publisher                Broker                  Subscriber
            │--- PUT /topic ---->│                          │
            │  protected with COSE│                          │
            │                    │<--- GET /topic ----- │
            │                    │                          │
            │                    │---- response ------> │
            │                    │  protected with COSE │
```

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

6.1. Using COSE Objects To Protect The Resource Representation

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

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

   The COSE object is constructed in the following way:

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

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

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

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

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

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

   The external_aad is an empty string.

   An example is given in Figure 12

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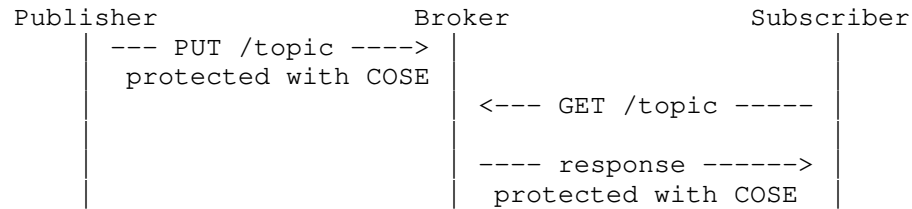        16(
          [
            / protected / h'a2010c04421234' / {
                \ alg \ 1:12,  \ AES-CCM-64-64-128 \
                \ kid \ 4: h'1234'
              } / ,
            / unprotected / {
              / iv / 5:h'89f52f65a1c580',
              / countersign / 7:[
                / protected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

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

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

7. Security Considerations

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

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

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

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

TODO: expand on security and privacy considerations

8. IANA Considerations

8.1. ACE Groupcomm Profile Registry

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

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

Name: coap_pubsub_app

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

CBOR Key: TBD

Reference: [[This document]]

8.2. ACE Groupcomm Key Registry

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

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

   Name: COSE_Key

   Key Type Value: TBD

   Profile: coap_pubsub_app

   Description: COSE_Key object

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

9. References

9.1. Normative References

   [I-D.ietf-ace-key-groupcomm]
             Palombini, F. and M. Tiloca, "Key Provisioning for Group
             Communication using ACE", draft-ietf-ace-key-groupcomm-03
             (work in progress), November 2019.

   [I-D.ietf-ace-oauth-authz]
             Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
             H. Tschofenig, "Authentication and Authorization for
             Constrained Environments (ACE) using the OAuth 2.0
             Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-25
             (work in progress), October 2019.

   [I-D.ietf-core-coap-pubsub]
             Koster, M., Keranen, A., and J. Jimenez, "Publish-
             Subscribe Broker for the Constrained Application Protocol
             (CoAP)", draft-ietf-core-coap-pubsub-09 (work in
             progress), September 2019.

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

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

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

9.2. Informative References

   [I-D.ietf-ace-actors]
               Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An
               architecture for authorization in constrained
               environments", draft-ietf-ace-actors-07 (work in
               progress), October 2018.

   [I-D.ietf-ace-dtls-authorize]
               Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and
               L. Seitz, "Datagram Transport Layer Security (DTLS)
               Profile for Authentication and Authorization for
               Constrained Environments (ACE)", draft-ietf-ace-dtls-
               authorize-08 (work in progress), April 2019.

   [I-D.ietf-ace-oscore-profile]
               Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson,
               "OSCORE profile of the Authentication and Authorization
               for Constrained Environments Framework", draft-ietf-ace-
               oscore-profile-08 (work in progress), July 2019.

Appendix A. Requirements on Application Profiles

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

o REQ18: Specify how the communication is secured between Client and
   KDC. Optionally, specify tranport profile of ACE
   [I-D.ietf-ace-oauth-authz] to use between Client and KDC: pre-set,
   as KDC is AS.

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

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

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

   o OPT4: Optionally, specify policies that instruct clients to retain
      unsuccessfully decrypted messages and for how long, so that they
      can be decrypted after getting updated keying material: not
      defined

Acknowledgments

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

Author's Address

   Francesca Palombini
   Ericsson

   Email: francesca.palombini@ericsson.com

              Key Provisioning for Group Communication using ACE
                   draft-palombini-ace-key-groupcomm-02

Abstract

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

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 April 25, 2019.

Copyright Notice

Table of Contents

1. Introduction

   This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to
   define the format of messages used to request, distribute and renew
   the keying material in a group communication scenario, e.g. based on
   multicast [RFC7390] or on publishing-subscribing
   [I-D.ietf-core-coap-pubsub].

   Profiles that use group communication can build on this document to
   specify the selection of the message parameters defined in this
   document to use and their values. Known applications that can
   benefit from this document would be, for example, profiles addressing
   group communication based on multicast [RFC7390] or publishing/
   subscribing [I-D.ietf-core-coap-pubsub] in ACE.

   If the application requires backward and forward security, updated
   keying material is generated and distributed to the group members
   (rekeying), when membership changes. A key management scheme
   performs the actual distribution of the updated keying material to

the group. In particular, the key management scheme rekeys the
current group members when a new node joins the group, and the
remaining group members when a node leaves the group. This document
provides a message format for group rekeying that allows to fulfill
these requirements. Rekeying mechanisms can be based on [RFC2093],
[RFC2094] and [RFC2627].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "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.

Readers are expected to be familiar with the terms and concepts
described in [I-D.ietf-ace-oauth-authz] and [RFC8152], such as
Authorization Server (AS) and Resource Server (RS).

2. Overview

```
+------------+                +----------+
|    AS      |                |   KDC    |
|            |      .-------->|          |
+------------+     /          +----------+
      ^           /
      |          /
      |         /
      v        /                              +----------+
+------------+ /            +------------+    |+----------+
|  Client    |<-'          | Dispatcher |    ||+----------+
|            |<-------->   |    (RS)    |<------->|  Group   |
+------------+             +------------+    +|  members |
                                             +----------+
```

Figure 1: Key Distribution Participants

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

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

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

o Key Distribution Center (KDC): maintains the keying material to
   protect group communications, and provides it to Clients

authorized to join the 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. If required by the application, the KDC renews
and re-distributes the keying material in the group when
membership changes.

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

This document specifies the message flows and formats for:

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

o Removing of a current member from the group (Section 5).

o Retrieving keying material as a current group member (Section 6
   and Section 7).

o Renewing and re-distributing the group keying material (rekeying)
   upon a membership change in the group (Section 4.2 and Section 5).

Figure 2 provides a high level overview of the message flow for a
node joining a group communication setting.

```
   C                                     AS   KDC   Dispatcher       Group
   |                                     |    |     |                Member
   |                                     |    |     | \              |
   |        Authorization Request        |    |     | | Defined      |
   |------------------------------------>|    |     | | in the ACE   |
   |                                     |    |     | | framework    |
   |        Authorization Response       |    |     | |              |
   |<------------------------------------|    |     | |              |
   |                                     |    |     | |              |
   |--------- Token Post --------------->|    |     | /              |
   |                                     |    |     |                |
   |---- Key Distribution Request ------>|    |     |                |
   |                                     |    |     |                |
   |<--- Key Distribution Response ------ | --- Group Rekeying ----->|
   |                                     |    |     |                |
   |<================== Protected communication ===|================>|
   |                                     |    |     |                |
```

                   Figure 2: Message Flow Upon New Node's Joining

   The exchange of Authorization Request and Authorization Response
   between Client and AS MUST be secured, as specified by the ACE
   profile used between Client and KDC.

   The exchange of Key Distribution Request and Key Distribution
   Response between Client and KDC MUST be secured, as a result of the
   ACE profile used between Client and KDC.

   All further communications between the Client and the KDC MUST be
   secured, for instance with the same security mechanism used for the
   Key Distribution exchange.

   All further communications between a Client and the other group
   members MUST be secured using the keying material provided in
   Section 4.

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 group. The first
   part of the exchange is based on ACE [I-D.ietf-ace-oauth-authz].

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

Section 3.2). Communications between the Client and the AS MUST be
secured, and depends on the profile of ACE used.

Figure 3 gives an overview of the exchange described above.

```
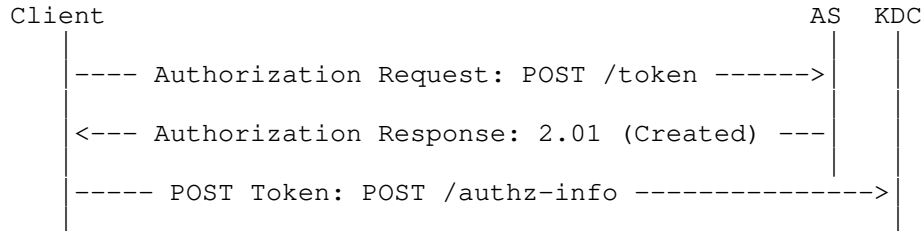     Client                                          AS  KDC
       |                                              |   |
       |---- Authorization Request: POST /token ----->|   |
       |                                              |   |
       |<--- Authorization Response: 2.01 (Created) ---|   |
       |                                              |   |
       |----- POST Token: POST /authz-info --------------->|
       |                                              |   |
```

            Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

   The Authorization Request sent from the Client to the AS is as
   defined in Section 5.6.1 of [I-D.ietf-ace-oauth-authz] and MUST
   contain the following parameters:

   o 'grant_type', with value "client_credentials".

   Additionally, the Authorization Request MAY contain the following
   parameters, which, if included, MUST have the corresponding values:

   o 'scope', with value the identifier of the specific group or topic
      the Client wishes to access, and optionally the role(s) the Client
      wishes to take. This value is a CBOR array encoded as a byte
      string, which contains:

      * As first element, the identifier of the specific group or
         topic.

      * Optionally, as second element, the role (or CBOR array of
         roles) the Client wishes to take in the group.

      The encoding of the group or topic identifier and of the role
      identifiers is application specific.

   o 'req_aud', as defined in Section 3.1 of
      [I-D.ietf-ace-oauth-params], with value an identifier of the KDC.

   o 'req_cnf', as defined in Section 3.1 of
      [I-D.ietf-ace-oauth-params], optionally containing the public key
      or the certificate of the Client, if it wishes to communicate that
      to the AS.

o Other additional parameters as defined in
   [I-D.ietf-ace-oauth-authz], if necessary.

3.2. Authorization Response

   The Authorization Response sent from the AS to the Client is as
   defined in Section 5.6.2 of [I-D.ietf-ace-oauth-authz] and MUST
   contain the following parameters:

   o 'access_token', containing the proof-of-possession access token.

   o 'cnf' if symmetric keys are used, not present if asymmetric keys
      are used. This parameter is defined in Section 3.2 of
      [I-D.ietf-ace-oauth-params] and contains the symmetric proof-of-
      possession key that the Client is supposed to use with the KDC.

   o 'rs_cnf' if asymmetric keys are used, not present if symmetric
      keys are used. This parameter is as defined in Section 3.2 of
      [I-D.ietf-ace-oauth-params] and contains information about the
      public key of the KDC.

   o 'exp', contains the lifetime in seconds of the access token. This
      parameter MAY be omitted if the application defines how the
      expiration time is communicated to the Client via other means, or
      if it establishes a default value.

   Additionally, the Authorization Response MAY contain the following
   parameters, which, if included, MUST have the corresponding values:

   o 'scope', which mirrors the 'scope' parameter in the Authorization
      Request (see Section 3.1). Its value is a CBOR array encoded as a
      byte string, containing:

      * As first element, the identifier of the specific group or topic
         the Client is authorized to access.

      * Optionally, as second element, the role (or CBOR array of
         roles) the Client is authorized to take in the group.

      The encoding of the group or topic identifier and of the role
      identifiers is application specific.

   o Other additional parameters as defined in
      [I-D.ietf-ace-oauth-authz], if necessary.

   The access token MUST contain all the parameters defined above
   (including the same 'scope' as in this message, if present, or the

'scope' of the Authorization Request otherwise), and additionally
other optional parameters the profile requires.

When receiving an Authorization Request from a Client that was
previously authorized, and which still owns a valid non expired
access token, the AS can simply reply with an Authorization Response
including a new access token.

3.3. Token Post

The Client sends a CoAP POST request including the access token to
the KDC, as specified in section 5.8.1 of [I-D.ietf-ace-oauth-authz].
If the specific ACE profile defines it, the Client MAY use a
different endpoint than /authz-info at the KDC to post the access
token to. After successful verification, the Client is authorized to
receive the group keying material from the KDC and join the group.

Note that this step could be merged with the following message from
the Client to the KDC, namely Key Distribution Request.

4. Key Distribution

This section defines how the keying material used for group
communication is distributed from the KDC to the Client, when joining
the group as a new member.

If not previously established, the Client and the KDC MUST first
establish a pairwise secure communication channel using ACE. The
exchange of Key Distribution Request-Response 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.

During this exchange, the Client sends a request to the AS,
specifying the group it wishes to join (see Section 4.1). Then, the
KDC verifies the access token and that the Client is authorized to
join that group; if so, it provides the Client with the keying
material to securely communicate with the member of the group (see
Section 4.2).

Figure 4 gives an overview of the exchange described above.

```
        Client                                             KDC
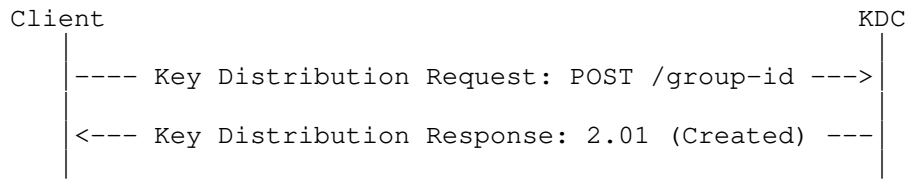          |                                                 |
          |---- Key Distribution Request: POST /group-id --->|
          |                                                 |
          |<--- Key Distribution Response: 2.01 (Created) ---|
          |                                                 |
```

        Figure 4: Message Flow of Key Distribution to a New Group Member

   The same set of message can also be used for the following cases,
   when the Client is already a group member:

   o The Client wishes to (re-)get the current keying material, for
      cases such as expiration, loss or suspected mismatch, due to e.g.
      reboot or missed group rekeying. This is further discussed in
      Section 6.

   o The Client wishes to (re-)get the public keys of other group
      members, e.g. if it is aware of new nodes joining the group after
      itself. This is further discussed in Section 7.

   Additionally, the format of the payload of the Key Distribution
   Response (Section 4.2) can be reused for messages sent by the KDC to
   distribute updated group keying material, in case of a new node
   joining the group or of a current member leaving the group. The key
   management scheme used to send such messages could rely on, e.g.,
   multicast in case of a new node joining or unicast in case of a node
   leaving the group.

   Note that 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.

4.1. Key Distribution Request

   The Client sends a Key Distribution request to the KDC. This
   corresponds to a CoAP POST request to the endpoint in the KDC
   associated to the group to join. The endpoint in the KDC is
   associated to the 'scope' value of the Authorization Request/
   Response. The payload of this request is a CBOR Map which MAY
   contain the following fields, which, if included, MUST have the
   corresponding values:

   o 'scope', with value the specific resource that the Client is
      authorized to access (i.e. group or topic identifier) and role(s),
      encoded as in Section 3.1.

   o 'get_pub_keys', if the Client wishes to receive the public keys of
      the other nodes in the group from the KDC. The value is an empty
      CBOR Array. This parameter may be present if the KDC stores the
      public keys of the nodes in the group and distributes them to the
      Client; it is useless to have here if the set of public keys of
      the members of the group is known in another way, e.g. it was
      provided by the AS.

   o 'client_cred', with value the public key or certificate of the
      Client. If the KDC is managing (collecting from/distributing to
      the Client) the public keys of the group members, this field
      contains the public key of the Client.

   o 'pub_keys_repos', can be present if a certificate is present in
      the 'client_cred' field, with value a list of public key
      repositories storing the certificate of the Client.

4.2. Key Distribution Response

   The KDC verifies the access token and, if verification succeeds,
   sends a Key Distribution success Response to the Client. This
   corresponds to a 2.01 Created message. The payload of this response
   is a CBOR Map which MUST contain the following fields:

   o 'key', used to send the keying material to the Client, as a
      COSE_Key ([RFC8152]) containing the following parameters:

      * 'kty', as defined in [RFC8152].

      * 'k', as defined in [RFC8152].

      * 'exp' (optionally), as defined below. This parameter is
         RECOMMENDED to be included in the COSE_Key. If omitted, the
         authorization server SHOULD provide the expiration time via
         other means or document the default value.

      * 'alg' (optionally), as defined in [RFC8152].

      * 'kid' (optionally), as defined in [RFC8152].

      * 'base iv' (optionally), as defined in [RFC8152].

      * 'clientID' (optionally), as defined in
         [I-D.ietf-ace-oscore-profile].

      * 'serverID' (optionally), as defined in
         [I-D.ietf-ace-oscore-profile].

    * 'kdf' (optionally), as defined in
       [I-D.ietf-ace-oscore-profile].

    * 'slt' (optionally), as defined in
       [I-D.ietf-ace-oscore-profile].

    * 'cs_alg' (optionally), containing the algorithm value to
       countersign the message, taken from Table 5 and 6 of [RFC8152].

The parameter 'exp' identifies the expiration time in seconds after
which the COSE_Key is not valid anymore for secure communication in
the group. A summary of 'exp' can be found in Figure 5.

```
+------+-------+----------------+-----------+----------------+
| Name | Label | CBOR Type      | Value     | Description    |
|      |       |                | Registry  |                |
+------+-------+----------------+-----------+----------------+
| exp  | TBD   | Integer or     | COSE Key  | Expiration time|
|      |       | floating-point | Common    | in seconds     |
|      |       | number         | Parameters|                |
+------+-------+----------------+-----------+----------------+
```

             Figure 5: COSE Key Common Header Parameter 'exp'

Optionally, the Key Distribution Response MAY contain the following
parameters, which, if included, MUST have the corresponding values:

o 'pub_keys', may only be present if 'get_pub_keys' was present in
   the Key Distribution Request; this parameter is a COSE_KeySet (see
   [RFC8152]), which contains the public keys of all the members of
   the group.

o 'group_policies', with value a list of parameters indicating how
   the group handles specific management aspects. This includes, for
   instance, approaches to achieve synchronization of sequence
   numbers among group members. The exact format of this parameter
   is specific to the profile.

o 'mgt_key_material', with value the administrative keying material
   to participate in the group rekeying performed by the KDC. The
   exact format and content depend on the specific rekeying scheme
   used in the group, which may be specified in the profile.

Specific profiles need to specify how exactly the keying material is
used to protect the group communication.

If the application requires backward security, the KDC SHALL generate
new group keying material and securely distribute it to all the

current group members, using the message format defined in this
section. Application profiles may define alternative message
formats.

TBD: define for verification failure

5. Removal of a Node from the Group

This section describes at a high level how a node can be removed from
the group.

If the application requires forward security, the KDC SHALL generate
new group keying material and securely distribute it to all the
current group members but the leaving node, using the message format
defined in Section 4.2. Application profiles may define alternative
message formats.

5.1. Expired Authorization

If the node is not authorized anymore, the AS can directly
communicate that to the KDC. Alternatively, the access token might
have expired. If Token introspection is provided by the AS, the KDC
can use it as per Section 5.7 of [I-D.ietf-ace-oauth-authz], in order
to verify that the access token is still valid.

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

5.2. Request to Leave the Group

A node can actively request to leave the group. In this case, the
Client can send a request formatted as follows to the KDC, to abandon
the group.

TBD: Format of the message to leave the group

The KDC should then remove the leaving node from the list of group
members, if the KDC keeps track of that.

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

6. Retrieval of Updated Keying Material

   A node stops using the group keying material upon its expiration,
   according to the 'exp' parameter specified in the retained COSE Key.
   Then, if it wants to continue participating in the group
   communication, the node has to request new updated keying material to
   the KDC.

   The Client may perform the same request to the KDC also upon
   receiving messages from other group members without being able to
   correctly decrypt them. This may be due to a previous update of the
   group keying material (rekeying) triggered by the KDC, that the
   Client was not able to receive or decrypt.

   Note that policies can be set up so that the Client sends a request
   to the KDC only after a given number of unsuccessfully decrypted
   incoming messages.

6.1. Key Re-Distribution Request

   To request a re-distribution of keying material, the Client sends a
   shortened Key Distribution Request to the KDC (Section 4.1),
   formatted as follows. The payload MUST contain only the following
   field:

   o 'scope', which contains only the identifier of the specific group
      or topic, encoded as in Section 3.1. That is, the role field is
      not present.

6.2. Key Re-Distribution Response

   The KDC receiving a Key Re-Distribution Request MUST check that it is
   storing a valid access token from that client for that scope.

   TODO: defines error response if it does not have it / is not valid.

   The KDC replies to the Client with a Key Distribution Response
   containing the 'key' parameter, and optionally 'group_policies' and
   'mgt_key_material', as specified in Section 4.2. Note that this
   response might simply re-provide the same keying material currently
   owned by the Client, if it has not been renewed.

7. Retrieval of Public Keys for Group Members

   In case the KDC maintains the public keys of group members, a node in
   the group can contact the KDC to request public keys of either all
   group members or a specified subset, using the messages defined
   below.

Figure 6 gives an overview of the exchange described above.

```
          Client                                      KDC
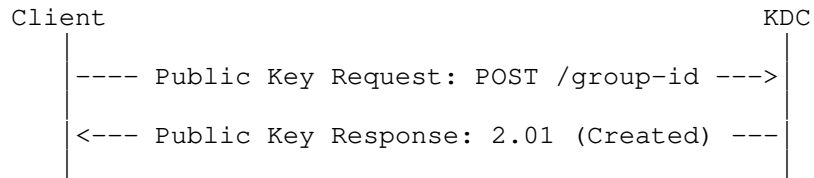            |                                          |
            |---- Public Key Request: POST /group-id --->|
            |                                          |
            |<--- Public Key Response: 2.01 (Created) ---|
            |                                          |
```

Figure 6: Message Flow of Public Key Request-Response

Note that these messages can be combined with the Key Re-Distribution messages in Section 6, to request at the same time the keying material and the public keys. In this case, either a new endpoint at the KDC may be used, or additional information needs to be sent in the request payload, to distinguish these combined messages from the Public Key messages described below, since they would be identical otherwise.

7.1. Public Key Request

To request public keys, the Client sends a shortened Key Distribution Request to the KDC (Section 4.1), formatted as follows. The payload of this request MUST contain the following fields:

o 'get_pub_keys', which has as value a CBOR array including either:

   * no elements, i.e. an empty array, in order to request the public key of all current group members; or

   * N elements, each of which is the identifier of a group member, in order to request the public key of the specified nodes.

o 'scope', which contains only the identifier of the specific group or topic, encoded as in Section 3.1. That is, the role field is not present.

7.2. Public Key Response

The KDC replies to the Client with a Key Distribution Response containing only the 'pub_keys' parameter, as specified in Section 4.2. The payload of this response contains the following field:

o 'pub_keys', which contains either:

* the public keys of all the members of the group, if the 'get_pub_keys' parameter of the Public Key request was an empty array; or

* the public keys of the group members with the identifiers specified in the 'get_pub_keys' parameter of the Public Key request.

The KDC ignores possible identifiers included in the 'get_pub_keys' parameter of the Public Key request if they are not associated to any current group member.

8. Security Considerations

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

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

9. IANA Considerations

The following registration is required for the COSE Key Common Parameter Registry specified in Section 16.5 of [RFC8152]:

o Name: exp

o Label: TBD

o CBOR Type: Integer or floating-point number

o Value Registry: COSE Key Common Parameters

o Description: Identifies the expiration time in seconds of the COSE Key

o Reference: [[this specification]]

10. References

10.1. 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)", draft-ietf-ace-oauth-authz-16
          (work in progress), October 2018.

[I-D.ietf-ace-oauth-params]
          Seitz, L., "Additional OAuth Parameters for Authorization
          in Constrained Environments (ACE)", draft-ietf-ace-oauth-
          params-00 (work in progress), September 2018.

[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-04 (work in progress), October 2018.

[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>.

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

10.2. Informative References

[I-D.ietf-core-coap-pubsub]
          Koster, M., Keranen, A., and J. Jimenez, "Publish-
          Subscribe Broker for the Constrained Application Protocol
          (CoAP)", draft-ietf-core-coap-pubsub-05 (work in
          progress), July 2018.

[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>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for
          the Constrained Application Protocol (CoAP)", RFC 7390,
          DOI 10.17487/RFC7390, October 2014,
          <https://www.rfc-editor.org/info/rfc7390>.

Acknowledgments

   The following individuals were helpful in shaping this document: Ben
   Kaduk, John Mattsson, Jim Schaad, Ludwig Seitz, Goeran Selander and
   Peter van der Stok.

Authors' Addresses

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

   Email: francesca.palombini@ericsson.com

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

   Email: marco.tiloca@ri.se

ACE Working Group   G. Selander
Internet-Draft   Ericsson AB
Intended status: Standards Track   S. Raza
Expires: 6 May 2021   RISE
                                                          M. Furuhed
                                                               Nexus
                                                          M. Vucinic
                                                           T. Claeys
                                                               INRIA
                                                     2 November 2020

                    Protecting EST Payloads with OSCORE
                    draft-selander-ace-coap-est-oscore-04

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 6 May 2021.

Copyright Notice

Table of Contents

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 [RFC7049] 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:

   * Compact representations of X.509 certificates (see
      [I-D.mattsson-cose-cbor-cert-compress])

   * Certificates by reference (see [I-D.ietf-cose-x509])

   * 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:

   * The DTLS record layer is replaced, or complemented, with OSCORE.

   * 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.

   * One new EST function, /rpks, is defined for installation of
      compact explicit TAs in the EST client.

   * 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

   * 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].

   * The certificates MAY be compressed, e.g. using the CBOR encoding
      defined in [I-D.mattsson-cose-cbor-cert-compress].

   * 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.

   * 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

   * The EST-oscore endpoints support delayed responses

   * The endpoints supports the following CoAP options: OSCORE, Uri-
      Host, Uri-Path, Uri-Port, Content-Format, Block1, Block2, and
      Accept.

   * 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.

```
                                        Constrained-Node Network
       .---------.                .---------------------------.
       |   CA    |                |.-------------------------.|
       ',_____'                ||                         ||
           |                      ||                         ||
       .------.     HTTP    .----------------.  CoAP   .-----------.||
       | EST  |<------->|  CoAP-to-HTTP  |<------->| EST Client|||
       |Server|   (TLS)  |     Proxy      |  (DTLS)  '-----------' ||
       '_____'          '_____'                       ||
                                 ||                                ||
        <------------------------------------------------->       ||
                      OSCORE     ||'-------------------------'|
                                 |'-------------------------'|
                                 '---------------------------'
```

          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

    [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>.

    [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
               Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
               October 2013, <https://www.rfc-editor.org/info/rfc7049>.

    [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>.

    [I-D.ietf-lake-edhoc]
               Selander, G., Mattsson, J., and F. Palombini, "Ephemeral
               Diffie-Hellman Over COSE (EDHOC)", Work in Progress,
               Internet-Draft, draft-ietf-lake-edhoc-01, 2 August 2020,
               <http://www.ietf.org/internet-drafts/draft-ietf-lake-
               edhoc-01.txt>.

   [I-D.ietf-ace-coap-est]
             Stok, P., Kampanakis, P., Richardson, M., and S. Raza,
             "EST over secure CoAP (EST-coaps)", Work in Progress,
             Internet-Draft, draft-ietf-ace-coap-est-18, 6 January
             2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-
             coap-est-18.txt>.

10.2. Informative References

   [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>.

   [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>.

   [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>.

   [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>.

   [I-D.ietf-6tisch-minimal-security]
              Vucinic, M., Simon, J., Pister, K., and M. Richardson,
              "Constrained Join Protocol (CoJP) for 6TiSCH", Work in
              Progress, Internet-Draft, draft-ietf-6tisch-minimal-
              security-15, 10 December 2019, <http://www.ietf.org/
              internet-drafts/draft-ietf-6tisch-minimal-security-
              15.txt>.

   [I-D.ietf-ace-oscore-profile]
              Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson,
              "OSCORE Profile of the Authentication and Authorization
              for Constrained Environments Framework", Work in Progress,
              Internet-Draft, draft-ietf-ace-oscore-profile-13, 27
              October 2020, <http://www.ietf.org/internet-drafts/draft-
              ietf-ace-oscore-profile-13.txt>.

   [I-D.ietf-ace-oauth-authz]
              Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
              H. Tschofenig, "Authentication and Authorization for
              Constrained Environments (ACE) using the OAuth 2.0
              Framework (ACE-OAuth)", Work in Progress, Internet-Draft,
              draft-ietf-ace-oauth-authz-35, 24 June 2020,
              <http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-
              authz-35.txt>.

   [I-D.ietf-core-oscore-groupcomm]
              Tiloca, M., Selander, G., Palombini, F., and J. Park,
              "Group OSCORE - Secure Group Communication for CoAP", Work
              in Progress, Internet-Draft, draft-ietf-core-oscore-
              groupcomm-09, 23 June 2020, <http://www.ietf.org/internet-
              drafts/draft-ietf-core-oscore-groupcomm-09.txt>.

   [I-D.ietf-cose-x509]
              Schaad, J., "CBOR Object Signing and Encryption (COSE):
              Header parameters for carrying and referencing X.509
              certificates", Work in Progress, Internet-Draft, draft-
              ietf-cose-x509-07, 17 September 2020,
              <http://www.ietf.org/internet-drafts/draft-ietf-cose-
              x509-07.txt>.

   [I-D.mattsson-cose-cbor-cert-compress]
              Raza, S., Hoglund, J., Selander, G., Mattsson, J., and M.
              Furuhed, "CBOR Profile of X.509 Certificates", Work in
              Progress, Internet-Draft, draft-mattsson-cose-cbor-cert-
              compress-01, 13 July 2020, <http://www.ietf.org/internet-
              drafts/draft-mattsson-cose-cbor-cert-compress-01.txt>.

   [I-D.palombini-core-oscore-edhoc]
              Palombini, F., Tiloca, M., Hoeglund, R., Hristozov, S.,
              and G. Selander, "Combining EDHOC and OSCORE", Work in
              Progress, Internet-Draft, draft-palombini-core-oscore-
              edhoc-00, 13 July 2020, <http://www.ietf.org/internet-
              drafts/draft-palombini-core-oscore-edhoc-00.txt>.

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).

```
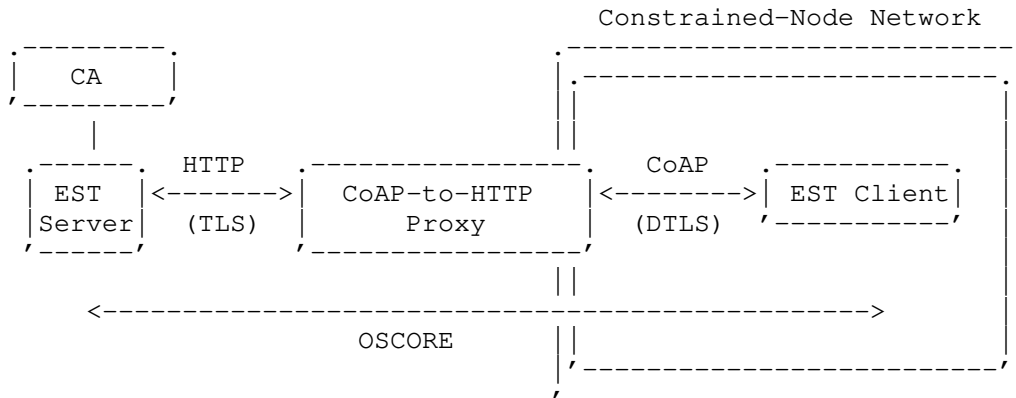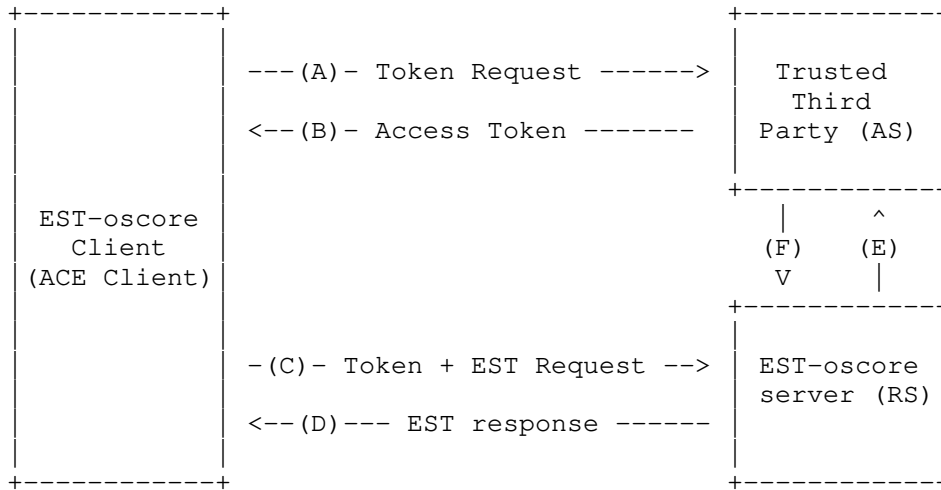       +-----------+                         +-----------+
       |           |                         |           |
       |           | ---(A)- Token Request ------> | Trusted   |
       |           |                         |   Third   |
       |           | <--(B)- Access Token ------- | Party (AS)|
       |           |                         |           |
       | EST-oscore|                         +-----------+
       |   Client  |                          |     ^
       |(ACE Client)|                        (F)    (E)
       |           |                          V     |
       |           |                         +-----------+
       |           |                         |           |
       |           | -(C)- Token + EST Request --> | EST-oscore|
       |           |                         | server (RS)|
       |           | <--(D)--- EST response ------ |           |
       |           |                         |           |
       +-----------+                         +-----------+
```

        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:

* an ID context distinguishing between different OSCORE security
   contexts to use,

* an AEAD algorithm,

* an HKDF algorithm,

* a master salt, and

* 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
   formated message [RFC2986]. The essential information fields for the
   CA to verify are the following:

   * Information about the subject, here condensed to the subject
      common name,

   * subject public key, and

   * 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'0123456789ABCDF0',
   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-----
   MIHcMIGEAgEAMCIxIDAeBgNVBAMMFzAxLTIzLTQ1LTY3LTg5LUFCLUNELUYwMFkw
   EwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEYeuA0qv31+QTnIa4fkJGbxtCINP3/51q
   GuKY+5rbtGSeZn3l8rVbU0jVEBWvKhAd98JeqgsuauGHRNWt2FqJ1aAAMAoGCCqG
   SM49BAMCA0cAMEQCIAZDSLnlLuDan5iE2N1BJIxJgEq5IzMOIIoWgXLcrhJ6AiBq
   BsBZV/HbjE4gdDe5q3c5y4V6pt2UhmJ7iWFgaitorg==
   -----END CERTIFICATE REQUEST-----

   In hex, 221 bytes:

   3081dc308184020100302231200301e06035504030c1730312d32332d34352d36
   372d38392d41422d43442d46303059301306072a8648ce3d020106082a8648ce
   3d0301070342000461eb80d2abf7d7e4139c86b87e42466f1b4220d3f7ff9d6a
   1ae298fb9adbb4649e667de5f2b55b5348d51015af2a101df7c25eaa0b2e6ae1
   8744d5add85a89d5a000300a06082a8648ce3d040302034700304402200643480
   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
INRIA

Email: timothy.claeys@inria.fr

ACE Working Group   M. Tiloca
Internet-Draft   RISE AB
Intended status: Standards Track   J. Park
Expires: April 25, 2019   Universitaet Duisburg-Essen
                                                            F. Palombini
                                                            Ericsson AB
                                                        October 22, 2018

Key Management for OSCORE Groups in ACE
draft-tiloca-ace-oscoap-joining-05

Abstract

   This document describes a method to request and provision keying
   material in group communication scenarios where communications are
   based on CoAP and secured with Object Security for Constrained
   RESTful Environments (OSCORE). The proposed method delegates the
   authentication and authorization of new client nodes that join an
   OSCORE group through a Group Manager server. This approach builds on
   the ACE framework for Authentication and Authorization, and leverages
   protocol-specific profiles of ACE to achieve communication security,
   proof-of-possession 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 April 25, 2019.

Table of Contents

1. Introduction

   Object Security for Constrained RESTful Environments (OSCORE)
   [I-D.ietf-core-object-security] is a method for application-layer
   protection of the Constrained Application Protocol (CoAP) [RFC7252],
   using CBOR Object Signing and Encryption (COSE) [RFC8152] and
   enabling end-to-end security of CoAP payload and options.

   As described in [I-D.ietf-core-oscore-groupcomm], OSCORE may be used
   to protect CoAP group communication over IP multicast [RFC7390].
   This relies on a Group Manager, which is responsible for managing an
   OSCORE group, where members exchange CoAP messages secured with
   OSCORE. The Group Manager can be responsible for multiple groups,
   coordinates the join process of new group members, and is entrusted
   with the distribution and renewal of group keying material.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines a method to:

o Authorize a node to join an OSCORE group, and provide it with the group keying material to communicate with other group members.

o Provide updated keying material to group members upon request.

o Renew the group keying material and distribute it to the OSCORE group (rekeying) upon changes in the group membership.

A client node joins an OSCORE group through a resource server acting as Group Manager for that group. The join process relies on an Access Token, which is bound to a proof-of-possession key and authorizes the client to access a specific join resource at the Group Manager.

Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm] for provisioning and renewing keying material in group communication scenarios.

In order to achieve communication security, proof-of-possession and server authentication, the client and the Group Manager leverage protocol-specific profiles of ACE. These include also possible forthcoming profiles that comply with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

1.1. Terminology

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

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

Readers are expected to be familiar with the terms and concepts related to the CoAP protocol described in [RFC7252][RFC7390]. Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS and /authz-info at the RS. This

document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

Readers are expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE [I-D.ietf-core-object-security] also in group communication scenarios [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications are secured with OSCORE. In this specification, the Group Manager acts as Resource Server.

This document refers also to the following terminology.

o Joining node: a network node intending to join an OSCORE group, where communication is based on CoAP [RFC7390] and secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].

o Join process: the process through which a joining node becomes a member of an OSCORE group. The join process is enforced and assisted by the Group Manager responsible for that group.

o Join resource: a resource hosted by the Group Manager, associated to an OSCORE group under that Group Manager. A join resource is identifiable with the Group Identifier (Gid) of the respective group. A joining node accesses a join resource to start the join process and become a member of that group.

o Join endpoint: an endpoint at the Group Manager associated to a join resource.

o Requester: member of an OSCORE group that sends request messages to other members of the group.

o Listener: member of an OSCORE group that receives request messages from other members of the group. A listener may reply back, by sending a response message to the requester which has sent the request message.

o Pure listener: member of a group that is configured as listener 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].

o Group rekeying process: the process through which the Group Manager renews the security parameters and group keying material, and (re-)distributes them to the OSCORE group members.

1.2. Relation to Other Documents

   Figure 1 overviews the main documents related to this specification.
   Arrows and asterisk-arrows denote normative references and
   informative refences, respectively.

```
                         +---------------------------------------+
                         |                                       |
           +---------------|-------------+                       |
           |               |             |                       |
           |               v             v              Key Management
        Pub-sub ---> Key Groupcomm ---> ACE Framework <--- for OSCORE Groups
        profile        *    *            [[WG]]            [[This document]]
           |           *    *             ^                      ^  |
           |           *    *             *                      *  |
           |           *    *             * ***************         |
           |  ***********    *            * *                      |
           |  *             *            * * +--------------+      |
     ACE   |  *             *            * * |              |      |
     ------|--*-------------*------------*-*-|--------------|-------
     CoRE  |  *             *            * * |              |
           v  v             v            * * v              v
           CoRE           CoRE         OSCORE -------------> OSCORE
          Pubsub        Groupcomm  <*** Groupcomm <************ [[WG]]
          [[WG]]        [[RFC7390]]     [[WG]]
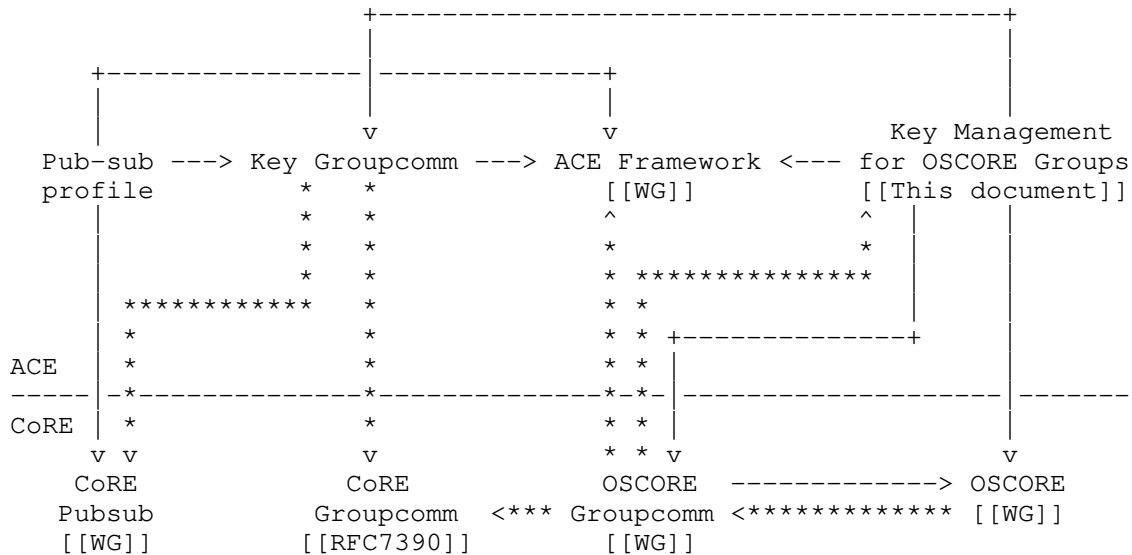```

                        Figure 1: Related Documents

2.  Protocol Overview

   Group communication for CoAP over IP multicast has been enabled in
   [RFC7390] and can be secured with Object Security for Constrained
   RESTful Environments (OSCORE) [I-D.ietf-core-object-security] as
   described 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 specification describes how to use the ACE framework for
   authentication and authorization [I-D.ietf-ace-oauth-authz] to:

   o  Enable a node to join an OSCORE group through the Group Manager
      and receive the security parameters and keying material to
      communicate with the other members of the gorup.

   o  Enable members of OSCORE groups to retrieve updated group keying
      material from the Group Manager.

o  Enable the Group Manager to renew the security parameters and
   group keying material, and to (re-)distribute them to the members
   of the OSCORE group (rekeying).

With reference to the ACE framework and the terminology defined in
OAuth 2.0 [RFC6749]:

o  The Group Manager acts as Resource Server (RS), and hosts one join
   resource for each OSCORE group it manages.  Each join resource is
   exported by a distinct join endpoint.  During the join process,
   the Group Manager provides joining nodes with the parameters and
   keying material for taking part to secure communications in the
   OSCORE group.  The Group Manager also maintains the group keying
   material and performs the group rekeying process to distribute
   updated keying material to the group members.

o  The joining node acts as Client (C), and requests to join an
   OSCORE group by accessing the related join endpoint at the Group
   Manager.

o  The Authorization Server (AS) authorizes joining nodes to join
   OSCORE groups under their respective Group Manager.  Multiple
   Group Managers can be associated to the same AS.  The AS MAY
   release Access Tokens for other purposes than joining OSCORE
   groups under registered Group Managers.  For example, the AS may
   also release Access Tokens for accessing resources hosted by
   members of OSCORE groups.

All communications between the involved entities rely on the CoAP
protocol and MUST be secured.

In particular, communications between the joining node and the Group
Manager leverage protocol-specific profiles of ACE to achieve
communication security, proof-of-possession and server
authentication.  To this end, the AS must signal the specific profile
to use, consistently with requirements and assumptions defined in the
ACE framework [I-D.ietf-ace-oauth-authz].

With reference to the AS, communications between the joining node and
the AS (/token endpoint) as well as between the Group Manager and the
AS (/introspect endpoint) can be secured by different means, for
instance using DTLS [RFC6347] or OSCORE
[I-D.ietf-core-object-security].  Further details on how the AS
secures communications (with the joining node and the Group Manager)
depend on the specifically used profile of ACE, and are out of the
scope of this specification.

2.1.  Overview of the Join Process

   A node performs the following steps in order to join an OSCORE group.
   Messages exchanged among the participants follow the formats defined
   in [I-D.palombini-ace-key-groupcomm], and are further specified in
   Section 3 and Section 4 of this document.  The Group Manager acts as
   the Key Distribution Center (KDC) defined in
   [I-D.palombini-ace-key-groupcomm].

   1.  The joining node requests an Access Token from the AS, in order
       to access a join resource on the Group Manager and hence join the
       associated OSCORE group (see Section 3).  The joining node will
       start or continue using a secure communication channel with the
       Group Manager, according to the response from the AS.

   2.  The joining node transfers authentication and authorization
       information to the Group Manager by posting the obtained Access
       Token (see Section 4).  After that, a joining node must have a
       secure communication channel established with the Group Manager,
       before starting to join an OSCORE group under that Group Manager
       (see Section 4).  Possible ways to provide a secure communication
       channel are DTLS [RFC6347] and OSCORE
       [I-D.ietf-core-object-security].

   3.  The joining node starts the join process to become a member of
       the OSCORE group, by accessing the related join resource hosted
       by the Group Manager (see Section 4).

   4.  At the end of the join process, the joining node has received
       from the Group Manager the parameters and keying material to
       securely communicate with the other members of the OSCORE group.

   5.  The joining node and the Group Manager maintain the secure
       channel, to support possible future communications.

   All further communications between the joining node and the Group
   Manager MUST be secured, for instance with the same secure channel
   mentioned in step 2.

2.2.  Overview of the Group Rekeying Process

   If the application requires backward and forward security, the Group
   Manager MUST generate new security parameters and group keying
   material, and distribute them to the group (rekeying) upon membership
   changes.

   That is, the group is rekeyed when a node joins the group as a new
   member, or after a current member leaves the group.  By doing so, a

joining node cannot access communications in the group prior its
joining, while a leaving node cannot access communications in the
group after its leaving.

Parameters and keying material include a new Group Identifier (Gid)
for the group and a new Master Secret for the OSCORE Common Security
Context of that group (see Section 2 of
[I-D.ietf-core-oscore-groupcomm]).

The Group Manager MUST support the Group Rekeying Process described
in Section 7.  Future application profiles may define alternative
message formats and distribution schemes to perform group rekeying.

3.  Joining Node to Authorization Server

This section describes how the joining node interacts with the AS in
order to be authorized to join an OSCORE group under a given Group
Manager.  In particular, it considers a joining node that intends to
contact that Group Manager for the first time.

The message exchange between the joining node and the AS consists of
the messages Authorization Request and Authorization Response defined
in Section 3 of [I-D.palombini-ace-key-groupcomm].

In case the specific AS associated to the Group Manager is unknown to
the joining node, the latter can rely on mechanisms like the
Unauthorized Resource Request message described in Section 5.1.1 of
[I-D.ietf-ace-oauth-authz] to discover the correct AS to contact.

3.1.  Authorization Request

The joining node contacts the AS, in order to request an Access Token
for accessing the join resource hosted by the Group Manager and
associated to the OSCORE group.  The Access Token request sent to the
/token endpoint follows the format of the Authorization Request
message defined in Section 3.1 of [I-D.palombini-ace-key-groupcomm].
In particular:

o  The 'scope' parameter MUST be present and MUST include:

   *  in the first element, either the Group Identifier (Gid) of the
      group to join under the Group Manager, or a value from which
      the Group Manager can derive the Gid of the group to join.  It
      is up to the application to define how the Group Manager
      possibly performs the derivation of the full Gid. Appendix C of
      [I-D.ietf-core-oscore-groupcomm] provides an example of
      structured Gid, composed of a fixed part, namely Group Prefix,
      and a variable part, namely Group Epoch.

        *  in the second element, the role(s) that the joining node
           intends to have in the group it intends to join.  Possible
           values are: "requester"; "listener"; and "pure listener".
           Possible combinations are: ["requester" , "listener"];
           ["requester" , "pure listener"].

     o  The 'req_aud' parameter MUST be present and is set to the
        identifier of the Group Manager.

3.2.  Authorization Response

   The AS is responsible for authorizing the joining node to join
   specific OSCORE groups, according to join policies enforced on behalf
   of the respective Group Manager.

   In case of successful authorization, the AS releases an Access Token
   bound to a proof-of-possession key associated to the joining node.

   Then, the AS provides the joining node with the Access Token as part
   of an Access Token response, which follows the format of the
   Authorization Response message defined in Section 3.2 of
   [I-D.palombini-ace-key-groupcomm].

   The 'exp' parameter MUST be present.  Other means for the AS to
   specify the lifetime of Access Tokens are out of the scope of this
   specification.

   The AS must include the 'scope' parameter in the response when the
   value included in the Access Token differs from the one specified by
   the joining node in the request.  In such a case, the second element
   of 'scope' MUST be present and includes the role(s) that the joining
   node is actually authorized to take in the group, encoded as
   specified in Section 3.1 of this document.

   Also, the 'profile' parameter indicates the specific profile of ACE
   to use for securing communications between the joining node and the
   Group Manager (see Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]).

   In particular, if symmetric keys are used, the AS generates a proof-
   of-possession key, binds it to the Access Token, and provides it to
   the joining node in the 'cnf' parameter of the Access Token response.
   Instead, if asymmetric keys are used, the joining node provides its
   own public key to the AS in the 'req_cnf' parameter of the Access
   Token request.  Then, the AS uses it as proof-of-possession key bound
   to the Access Token, and provides the joining node with the Group
   Manager's public key in the 'rs_cnf' parameter of the Access Token
   response.

4.  Joining Node to Group Manager

   First, the joining node posts the Access Token to the /authz-info
   endpoint at the Group Manager, in accordance with the Token post
   defined in Section 3.3 of [I-D.palombini-ace-key-groupcomm].  Then,
   the joining node establishes a secure channel with the Group Manager,
   according to what is specified in the Access Token response and to
   the signalled profile of ACE.

4.1.  Join Request

   Once a secure communication channel with the Group Manager has been
   established, the joining node requests to join the OSCORE group, by
   accessing the related join resource at the Group Manager.

   In particular, the joining node sends to the Group Manager a
   confirmable CoAP request, using the method POST and targeting the
   join endpoint associated to that group.  This join request follows
   the format and processing of the Key Distribution Request message
   defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm].  In
   particular:

   o  The 'get_pub_keys' parameter is present only if the joining node
      wants to retrieve the public keys of the group members from the
      Group Manager during the join process (see Section 6).  Otherwise,
      this parameter MUST NOT be present.

   o  The 'client_cred' parameter, if present, includes the public key
      of the joining node.  This parameter MAY be omitted if: i) public
      keys are used as proof-of-possession keys between the joining node
      and the Group Manager; or ii) the joining node is asking to access
      the group exclusively as pure listener; or iii) the Group Manager
      already acquired this information during a previous join process.
      In any other case, this parameter MUST be present.

4.2.  Join Response

   The Group Manager processes the request according to
   [I-D.ietf-ace-oauth-authz].  If this yields a positive outcome, the
   Group Manager updates the group membership by registering the joining
   node as a new member of the OSCORE group.

   The Group Manager replies to the joining node providing the updated
   security parameters and keying meterial necessary to participate in
   the group communication.  This join response follows the format and
   processing of the Key Distribution success Response message defined
   in Section 4.2 of [I-D.palombini-ace-key-groupcomm].  In particular:

   o  The 'key' parameter includes what the joining node needs in order
      to set up the OSCORE Security Context as per Section 2 of
      [I-D.ietf-core-oscore-groupcomm].  In particular:

      *  The 'kty' parameter has value "Symmetric".

      *  The 'k' parameter includes the OSCORE Master Secret.

      *  The 'exp' parameter specifies when the OSCORE Security Context
         derived from these parameters expires.

      *  The 'alg' parameter, if present, has as value the AEAD
         algorithm used in the group.

      *  The 'kid' parameter, if present, has as value the identifier of
         the key in the parameter 'k'.

      *  The 'base IV' parameter, if present, has as value the OSCORE
         Common IV.

      *  The 'clientID' parameter, if present, has as value the OSCORE
         Sender ID assigned to the joining node by the Group Manager.
         This parameter is not present if the node joins the group
         exclusively as pure listener, according to what specified in
         the Access Token (see Section 3.2).  In any other case, this
         parameter MUST be present.

      *  The 'serverID' parameter MUST be present and has as value the
         Group Identifier (Gid) associated to the group.

      *  The 'kdf' parameter, if present, has as value the KDF algorithm
         used in the group.

      *  The 'slt' parameter, if present, has as value the OSCORE Master
         Salt.

      *  The 'cs_alg' parameter MUST be present and has as value the
         countersignature algorithm used in the group.

   o  The 'pub_keys' parameter is present only if the 'get_pub_keys'
      parameter was present in the join request.  If present, this
      parameter includes the public keys of the group members that are
      relevant to the joining node.  That is, it includes: i) the public
      keys of the non-pure listeners currently in the group, in case the
      joining node is configured (also) as requester; and ii) the public
      keys of the requesters currently in the group, in case the joining
      node is configured (also) as listener or pure listener.

o  The 'group_policies' parameter SHOULD be present and includes a
   list of parameters indicating particular policies enforced in the
   group.  For instance, it can indicate the method to achieve
   synchronization of sequence numbers among group members (see
   Appendix E of [I-D.ietf-core-oscore-groupcomm]).

Finally, the joining node uses the information received in the join
response to set up the OSCORE Security Context, as described in
Section 2 of [I-D.ietf-core-oscore-groupcomm].  From then on, the
joining node can exchange group messages secured with OSCORE as
described in [I-D.ietf-core-oscore-groupcomm].

If the application requires backward security, the Group Manager
SHALL generate updated security parameters and group keying material,
and provide it to all the current group members (see Section 7).

When the OSCORE Master Secret expires, as specified by 'exp' in the
'key' parameter of the join response, the node considers the OSCORE
Security Context also invalid and to be renewed.  Then, the node
retrieves updated security parameters and keying material, by
exchanging shortened Join Request and Join Response messages with the
Group Manager, according to the approach defined in Section 6 of
[I-D.palombini-ace-key-groupcomm].  Finally, the node uses the
updated security parameters and keying material to set up the new
OSCORE Security Context as described in Section 2 of
[I-D.ietf-core-oscore-groupcomm].

5.  Leaving of a Group Member

   A node may be removed from the OSCORE group, due to expired or
   revoked authorization, or after its own request to the Group Manager.

   If the application requires forward security, the Group Manager SHALL
   generate updated security parameters and group keying material, and
   provide it to the remaining group members (see Section 7).  The
   leaving node must not be able to acquire the new security parameters
   and group keying material distributed after its leaving.

   Same considerations in Section 5 of [I-D.palombini-ace-key-groupcomm]
   apply here as well, considering the Group Manager acting as KDC.  In
   particular, a node requests to leave the OSCORE group as described in
   Section 5.2 of [I-D.palombini-ace-key-groupcomm].

6.  Public Keys of Joining Nodes

   Source authentication of OSCORE messages exchanged within the group
   is ensured by means of digital counter signatures (see Sections 2 and
   3 of [I-D.ietf-core-oscore-groupcomm]).  Therefore, group members

must be able to retrieve each other's public key from a trusted key
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 public keys of the group
members, and provides those public keys to group members if requested
to.  Upon joining an OSCORE group, a joining node is thus expected to
provide its own public key to the Group Manager.

In particular, four cases can occur when a new node joins a group.

o  The joining node is going to join the group exclusively as pure
   listener.  That is, it is not going to send messages to the group,
   and hence to produce signatures with its own private key.  In this
   case, the joining node is not required to provide its own public
   key to the Group Manager upon joining the group.

o  The Group Manager already acquired the public key of the joining
   node during a previous join process.  In this case, the joining
   node may not provide again its own public key to the Group
   Manager, in order to limit the size of the join request.

o  The joining node and the Group Manager use an asymmetric proof-of-
   possession key to establish a secure communication channel.  In
   this case, the Group Manager stores the proof-of-possession key
   conveyed in the Access Token as the public key of the joining
   node.

o  The joining node and the Group Manager use a symmetric proof-of-
   possession key to establish a secure communication channel.  In
   this case, upon performing a join process with that Group Manager
   for the first time, the joining node specifies its own public key
   in the 'client_cred' parameter of the join request targeting the
   join endpoint (see Section 4.1).

Furthermore, as described in Section 4.1, the joining node may have
explicitly requested the Group Manager to retrieve the public keys of
the current group members, i.e. through the 'get_pub_keys' parameter
in the join request.  In this case, the Group Manager includes also
such public keys in the 'pub_keys' parameter of the join response
(see Section 4.2).

Later on as a group member, the node may need to retrieve the public
keys of other group members.  The node can do that by exchanging
shortened Join Request and Join Response messages with the Group
Manager, according to the approach defined in Section 7 of
[I-D.palombini-ace-key-groupcomm].

7.  Group Rekeying Process

   In order to rekey the OSCORE group, the Group Manager distributes a
   new Group ID of the group and a new OSCORE Master Secret for that
   group.  To this end, the Group Manager MUST support at least the
   following group rekeying scheme.  Future application profiles may
   define alternative message formats and distribution schemes.

   The Group Manager uses the same format of the Join Response message
   in Section 4.2.  In particular:

   o  Only the 'key' parameter is present.

   o  The 'k' parameter of the 'key' parameter includes the new OSCORE
      Master Secret.

   o  The 'serverID' parameter of the 'key' parameter includes the new
      Group ID.

   The Group Manager separately sends a group rekeying message to each
   group member to be rekeyed.  Each rekeying message MUST be secured
   with the pairwise secure communication channel between the Group
   Manager and the group member used during the join process.

8.  Security Considerations

   The method described in this document leverages the following
   management aspects related to OSCORE groups and discussed in the
   sections of [I-D.ietf-core-oscore-groupcomm] referred below.

   o  Management of group keying material (see Section 2.1 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).  According to
      the specific application requirements, this can include rekeying
      the group upon changes in its membership.  In particular, renewing
      the keying material is required upon a new node's joining or a
      current node's leaving, in case backward security and forward
      security have to be preserved, respectively.

   o  Provisioning and retrieval of public keys (see Section 2 of
      [I-D.ietf-core-oscore-groupcomm]).  The Group Manager acts as key
      repository of public keys of group members, and provides them upon
      request.

   o  Synchronization of sequence numbers (see Section 5 of
      [I-D.ietf-core-oscore-groupcomm]).  This concerns how a listener

node that has just joined an OSCORE group can synchronize with the
sequence number of requesters in the same group.

Before sending the join response, the Group Manager should verify
that the joining node actually owns the associated private key, for
instance by performing a proof-of-possession challenge-response,
whose details are out of the scope of this specification.

Further security considerations are inherited from
[I-D.palombini-ace-key-groupcomm], the ACE framework for
Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the
specific profile of ACE signalled by the AS, such as
[I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

9.  IANA Considerations

   This document has no actions for IANA.

10.  References

10.1.  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)", draft-ietf-ace-oauth-authz-16
             (work in progress), October 2018.

   [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-04 (work in progress), October 2018.

   [I-D.ietf-core-object-security]
             Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
             "Object Security for Constrained RESTful Environments
             (OSCORE)", draft-ietf-core-object-security-15 (work in
             progress), August 2018.

   [I-D.ietf-core-oscore-groupcomm]
             Tiloca, M., Selander, G., Palombini, F., and J. Park,
             "Secure group communication for CoAP", draft-ietf-core-
             oscore-groupcomm-02 (work in progress), June 2018.

   [I-D.palombini-ace-key-groupcomm]
             Palombini, F. and M. Tiloca, "Key Provisioning for Group
             Communication using ACE", draft-palombini-ace-key-
             groupcomm-02 (work in progress), October 2018.

   [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>.

   [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>.

10.2.  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)", draft-ietf-ace-dtls-
             authorize-05 (work in progress), October 2018.

   [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>.

   [RFC7390]  Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for
             the Constrained Application Protocol (CoAP)", RFC 7390,
             DOI 10.17487/RFC7390, October 2014,
             <https://www.rfc-editor.org/info/rfc7390>.

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

Acknowledgments

Authors' Addresses

   Marco Tiloca
   RISE AB
   Isafjordsgatan 22
   Kista  SE-164 29 Stockholm
   Sweden

   Email: marco.tiloca@ri.se


   Jiye Park
   Universitaet Duisburg-Essen
   Schuetzenbahn 70
   Essen  45127
   Germany

   Email: ji-ye.park@uni-due.de


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

   Email: francesca.palombini@ericsson.com