ACE Working Group   L. Seitz
Internet-Draft   SICS
Intended status: Standards Track   G. Selander
Expires: May 4, 2017   Ericsson
                                                          E. Wahlstroem

                                                           S. Erdtman
                                                           Spotify AB
                                                         H. Tschofenig
                                                             ARM Ltd.
                                                     October 31, 2016

Authentication and Authorization for Constrained Environments (ACE)
draft-ietf-ace-oauth-authz-04

Abstract

   This specification defines a framework for authentication and
   authorization in Internet of Things (IoT) environments. The
   framework is based on a set of building blocks including OAuth 2.0
   and CoAP, thus making a well-known and widely used authorization
   solution suitable for IoT devices. Existing specifications are used
   where possible, but where the constraints of IoT devices require it,
   extensions are added and profiles are defined.

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 http://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 4, 2017.

Copyright Notice

Table of Contents

1. Introduction

   Authorization is the process for granting approval to an entity to
   access a 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 is a complex
   task.

   While prior work on authorization solutions for the Web and for the
   mobile environment also applies to the 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 makes use of CoAP [RFC7252].

   A detailed treatment of constraints can be found in [RFC7228], and
   the different IoT deployments present 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 profiles.
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.

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].

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

Since we describe exchanges as RESTful protocol interactions HTTP
[RFC7231] offers useful terminology.

Terminology for entities in the architecture is defined in OAuth 2.0
[RFC6749] and [I-D.ietf-ace-actors], 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
/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.

Since this specification focuses on the problem of access control to
resources, we simplify the actors by assuming that the client
authorization server (CAS) functionality is not stand-alone but
subsumed by either the authorization server or the client (see
section 2.2 in [I-D.ietf-ace-actors]).

We call the specifications of this memo the "framework" or "ACE
framework". When referring to "profiles of this framework" we mean
additional memo's that define the use of this specification with
concrete transport, and communication security protocols (e.g. CoAP
over DTLS).

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, we do envision further
underlying protocols to be supported in the future, such as HTTP/2,
MQTT and QUIC.

A third building block is CBOR [RFC7049] for encodings where JSON
[RFC7159] 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 CoAP POST
parameters and CoAP responses.

A fourth building block is the compact CBOR-based secure message
format COSE [I-D.ietf-cose-msg], which enables application layer
security as an alternative or complement to transport layer security
(DTLS [RFC6347] or TLS [RFC5246]). COSE is used to secure self
contained tokens such as proof-of-possession (PoP) tokens, which is
an extension to the OAuth access tokens, and "client tokens" which
are defined in this framework (see Section 7.4). The default access
token format is defined in CBOR web token (CWT)
[I-D.ietf-ace-cbor-web-token]. Application layer security for CoAP
using COSE can be provided with OSCOAP
[I-D.selander-ace-object-security].

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 RFC 7228 [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. We believe this 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
form of ACE profiles.

In the subsections below we provide further details about the
different building blocks.

3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain
limited 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 introspect 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).

   The token introspection endpoint, /introspect, is used by the RS
   when requesting additional information regarding a received access
   token. The RS makes a POST request to /introspect 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 authorization server and consumed by
   the resource server. 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.

Proof of Possession Tokens:

An access token may be bound to a cryptographic key, which is then
used by an RS to authenticate requests from a client. Such tokens
are called proof-of-possession tokens (or PoP tokens).

The proof-of-possession (PoP) security concept assumes that the AS
acts as a trusted third party that binds keys to 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 token unless specifically stated
otherwise.

The key bound to the access token (aka PoP key) may be based on
symmetric as well as on asymmetric cryptography. The appropriate
choice of security 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 access token. The PoP key is
   also encrypted for the client and sent together with the access
   token to the client.

Asymmetric PoP key: An asymmetric key pair is generated on the
   client and the public key is sent to the AS (if it does not
   already have knowledge of the client's public key).
   Information about the public key, which is the PoP key in this
   case, is either stored to be returned on introspection calls or
   included inside the access token and sent back to the
   requesting client. The RS can identify the client's public key
   from the information in the token, which allows the client to
   use the corresponding private key for the proof of possession.

The access token is either a simple reference, or a structured
information object (e.g. CWT [I-D.ietf-ace-cbor-web-token]),
protected by a cryptographic wrapper (e.g. COSE
[I-D.ietf-cose-msg]). 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
   uses CBOR encoded messages for CoAP, defined in Section 5, to
   request scopes and to be informed what scopes the access token was
   actually authorized for by the AS.

   The values of the scope parameter 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 type-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 [I-D.ietf-ace-cbor-web-token] 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 a 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 need 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 in so-
called 'options'.

CoAP supports application-layer fragmentation of the CoAP payloads
through blockwise transfers [RFC7959]. However, block-wise 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 1.2
[RFC6347] or TLS 1.2 [RFC5246]. CoAP defines a number of proxy
operations which requires 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 on
application layer using an object-based security mechanism such as
COSE [I-D.ietf-cose-msg].

One application of COSE is OSCOAP [I-D.selander-ace-object-security],
which provides end-to-end confidentiality, integrity and replay
protection, and a secure binding between CoAP request and response
messages. In OSCOAP, the CoAP messages are wrapped in COSE objects
and sent using CoAP.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions
using the /token and /introspect endpoints. A client obtains an
access token from an AS using the /token endpoint and subsequently
presents the access token to a RS to gain access to a protected
resource. The RS, after receiving an access token, may present it to
the AS via the /introspect endpoint to get information about the
access token. In other deployments the RS may process the access
token locally without the need to contact an AS. 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 RFC 7521 [RFC7521] and
[I-D.ietf-oauth-device-flow]). What grant types works best depends
on the usage scenario and RFC 7744 [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 RFC 7521) and
the Client Credentials Grant (described in Section 4.4 of RFC 7521).

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
[I-D.ietf-oauth-native-apps] 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 or another person on his or her behalf have arranged with the
authorization server out-of-band, 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. We
assume 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
registration procedure implies that the client and the AS share
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. The
detailed procedures for this discovery process may be defined in an

ACE profile and depend on the protocols being used and the specific
deployment environment.

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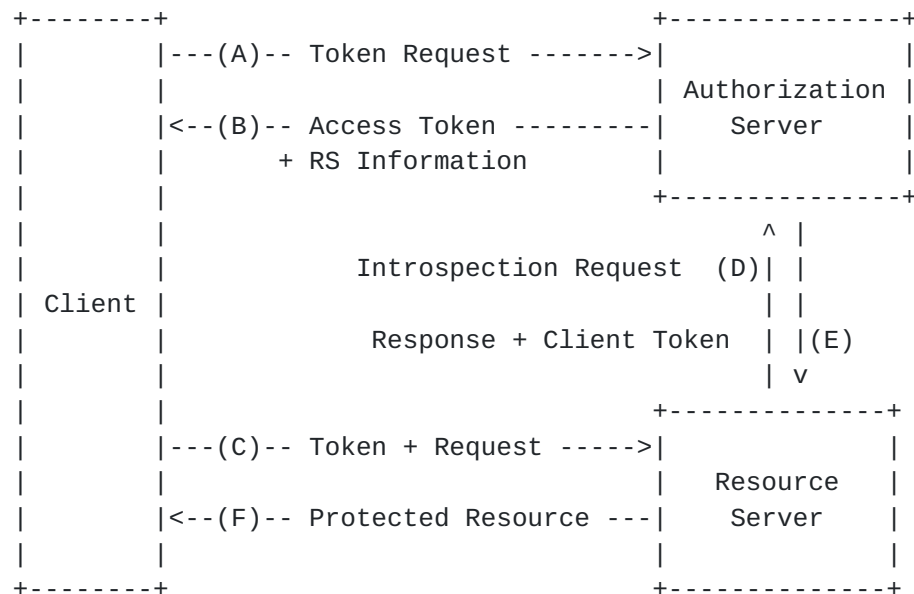
                   Figure 1: Basic Protocol Flow.

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 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. It also returns various parameters,
   referred as "RS Information". In addition to the response
   parameters defined by OAuth 2.0 and the PoP token extension,
   further response parameters, such as information on which profile
   the client should use with the resource server(s). More
   information about these parameters can be found in Section 6.4.

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 RS 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 RS Information or the client
   token. The RS verifies that the token is integrity protected by
   the AS and 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, in which case the different parts of
   step C may be interleaved with introspection.

Token Introspection Request (D):

   A resource server may be configured to introspect the access token
   by including it in a request to the /introspect endpoint at that
   AS. Token introspection over CoAP is defined in Section 7 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. The AS can additionally
   return information that the RS needs to pass on to the client in
   the form of a client token. The latter is used to establish keys

for mutual authentication between client and RS, when the client
has no direct connectivity to the AS, see Section 7.4 for details.

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 used communication security protocol.

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 we cannot generally assume that the client and RS are part
of a common key infrastructure, so the AS provisions credentials
or associated information to allow mutual authentication. These
credentials need to be provided to the parties before or during
the authentication protocol is executed, and may be re-used for
subsequent token requests.

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 indicating what key is used for mutual authentication.
If clients need to update a token, e.g. to get additional rights,
they can request that the AS binds the new access token to the
same credential as the previous token.

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
"profile" parameter in the token request and 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 for

   introspection. In constrained settings TLS is not always feasible,
   or desirable. Nevertheless it is REQUIRED that the data exchanged
   with the AS is encrypted and integrity protected. It is furthermore
   REQUIRED that the AS and the endpoint communicating with it (client
   or RS) perform mutual authentication.

   Profiles are expected to specify the details of how this is done,
   depending e.g. on the communication protocol and the credentials used
   by the client or the RS.

   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. This framework RECOMMENDS to use CoAP instead and
   RECOMMENDS the 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. The Content-format depends on the security applied to the
   content and must be specified by the corresponding profile.

   The OAuth 2.0 AS uses a JSON structure in the payload of its
   responses both to client and RS. This framework RECOMMENDS the use
   of CBOR [RFC7049] instead. The requesting device can explicitly
   request this encoding by setting the CoAP Accept option in the
   request to "application/cbor". Depending on the profile, the content
   may arrive in a different format wrapping a CBOR payload.

6. The 'Token' Endpoint

   In plain 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 client and
   RS to establish shared keys or to exchange their public keys.
   Furthermore this framework defines encodings using CoAP and CBOR,
   instead of HTTP and JSON.

   Communication between the client and the /token endpoint at the AS
   MUST be integrity protected and encrypted. Furthermore AS and client
   MUST perform mutual authentication. Profiles of this framework are
   expected to specify how authentication and communication security is
   implemented.

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

6.1. Client-to-AS Request

   The client sends a CoAP POST request to the token endpoint at the AS,
   the profile is expected to specify the Content-Type and wrapping of
   the payload. The content of the request consists of the parameters
   specified in section 4 of the OAuth 2.0 specification [RFC6749]
   encoded as a CBOR map.

   In addition to these parameters, this framework defines the following
   parameters for requesting an access token from a /token endpoint:

   aud
      OPTIONAL. Specifies the audience for which the client is
      requesting an access token. If this parameter is missing it is
      assumed that the client and the AS have a pre-established
      understanding of the audience that an access token should address.
      If a client submits a request for an access token without
      specifying an "aud" parameter, and the AS does not have a default
      "aud" value for this client, then the AS MUST respond with an
      error message with the CoAP response code 4.00 (Bad Request).

   cnf
      OPTIONAL. This field contains information about the key the
      client would like to bind to the access token for proof-of-
      possession. It is NOT RECOMMENDED that a client submits a
      symmetric key value to the AS using this parameter. See
      Section 6.4.5 for more details on the formatting of the 'cnf'
      parameter.

   The following examples illustrate different types of requests for
   proof-of-possession tokens.

   Figure 2 shows a request for a token with a symmetric proof-of-
   possession key. Note that in this example we assume a DTLS-based
   communication security profile, therefore the Content-Type is
   "application/cbor".

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "aud" : "tempSensor4711",
 }
```

   Figure 2: Example request for an access token bound to a symmetric
                                 key.

   Figure 3 shows a request for a token with an asymmetric proof-of-
   possession key. Note that in this example we assume an object
   security-based profile, therefore the Content-Type is "application/
   cose+cbor".

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cose+cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h'11',
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
    }
  }
}
```

   Figure 3: Example request for an access token bound to an asymmetric
                                 key.

   Figure 4 shows a request for a token where a previously communicated
   proof-of-possession key is only referenced. Note that we assume a
   DTLS-based communication security profile for this example, therefore
   the Content-Type is "application/cbor". Also note that the client
   performs a password based authentication in this example by
   submitting its client_secret (see section 2.3.1. of [RFC6749]).

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id" : "myclient",
  "client_secret" : "mysecret234",
  "aud" : "valve424",
  "scope" : "read",
  "cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

        Figure 4: Example request for an access token bound to a key
                              reference.

6.2. 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 CoAP
   response code 2.01 (Created). If client request was invalid, or not
   authorized, the AS returns an error response as described in
   Section 6.3.

   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. This prior
   knowledge may, for example, be set by the use of a dynamic client
   registration protocol exchange [RFC7591].

   The content of the successful reply MUST be encoded as CBOR map,
   containing parameters as speficied in section 5.1 of [RFC6749]. In
   addition to these parameters, the following parameters are also part
   of a successful response:

   profile
      REQUIRED. This indicates the profile that the client MUST use
      towards the RS. See Section 6.4.4 for the formatting of this
      parameter.

   cnf
      REQUIRED if the token type is 'pop'. OPTIONAL otherwise. If a
      symmetric proof-of-possession algorithms was selected, this field
      contains the proof-of-possession key. If an asymmetric algorithm

was selected, this field contains information about the public key
used by the RS to authenticate. See Section 6.4.5 for the
formatting of this parameter.
token_type
   OPTIONAL. 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.

Note that if CBOR Web Tokens [I-D.ietf-ace-cbor-web-token] are used,
the access token can also contain a 'cnf' claim. This claim is
however consumed by a different party. The access token is created
by the AS and processed by the RS (and opaque to the client) whereas
the RS Information is created by the AS and processed by the client;
it is never forwarded to the resource server.

The following examples illustrate different types of responses for
proof-of-possession tokens.

Figure 5 shows a response containing a token and a 'cnf' parameter
with a symmetric proof-of-possession key. Note that we assume a
DTLS-based communication security profile for this example, therefore
the Content-Type is "application/cbor".

```
Header: Created (Code=2.01)
Content-Type: "application/cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (remainder of CWT omitted for brevity;
   CWT contains COSE_Key in the 'cnf' claim)',
  "profile" : "coap_dtls",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 5: Example AS response with an access token bound to a
symmetric key.

6.3. 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 5.2 of [RFC6749], with the following differences: The
Content-Type is specified by the communication security profile used
between client and AS. The raw payload before being processed by the
communication security protocol MUST be encoded as a CBOR map and the
CoAP response code 4.00 (Bad Request) MUST be used unless specified
otherwise.

6.4. New 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.

6.4.1. Audience

This parameter specifies for which audience the client is requesting
a token. It should be encoded as CBOR text string (major type 3).
The formatting and semantics of these strings are application
specific.

6.4.2. Grant Type

The abbreviations in Figure 6 MAY be used in CBOR encodings instead
of the string values defined in [RFC6749].

```
       /--------------------+----------+--------------\
       | grant_type         | CBOR Key | Major Type   |
       |--------------------+----------+--------------|
       | password           |    0     |    0 (uint)  |
       | authorization_code |    1     |    0         |
       | client_credentials |    2     |    0         |
       | refresh_token      |    3     |    0         |
       \--------------------+----------+--------------/
```

Figure 6: CBOR abbreviations for common grant types

6.4.3. Token Type

The 'token_type' parameter allows the AS to indicate to the client
which type of access token it is receiving (e.g. a bearer token).
The 'pop' token type MUST be assumed by default if the AS does not
provide a different value.

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 is performed is specified by the profiles.

The values in the 'token_type' parameter are CBOR text strings (major
type 3).

6.4.4. Profile

Profiles of this framework are expected to define the communication
protocol and the communication security protocol between the client
and the RS. Furthermore profiles are expected to define proof-of-
possession methods, if they support proof-of-possession tokens.

A profile should specify an identifier that is used to uniquely
identify itself in the 'profile' parameter.

Profiles MAY define additional parameters for both the token request
and the RS Information in the access token response in order to
support negotioation or signalling of profile specific parameters.

6.4.5. Confirmation

The "cnf" parameter identifies or provides the key used for proof-of-
possession or for authenticating the RS depending on the proof-of-
possession algorithm and the context cnf is used in. This framework
extends the definition of 'cnf' from [RFC7800] by adding CBOR/COSE
encodings and the use of 'cnf' for transporting keys in the RS
Information.

The "cnf" parameter is used in the following contexts with the
following meaning:

o In the access token, to indicate the proof-of-possession key bound
   to this token.
o In the token request C -> AS, to indicate the client's raw public
   key, or the key-identifier of a previously established key between
   C and RS.
o In the token response AS -> C, to indicate either the symmetric
   key generated by the AS for proof-of-possession or the raw public
   key used by the RS to authenticate.
o In the introspection response AS -> RS, to indicate the proof-of-
   possession key bound to the introspected token.
o In the client token AS -> RS -> C, to indicate the proof-of-
   possession key bound to the access token.

A CBOR encoded payload MAY contain the 'cnf' parameter with the
following contents:

COSE_Key In this case the 'cnf' parameter contains the proof-of-
   possession key to be used by the client. An example is shown in
   Figure 7.

```
"cnf" : {
  "COSE_Key" : {
    "kty" : "EC",
    "kid" : h'11',
    "crv" : "P-256",
    "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
    "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
  }
}
```

         Figure 7: Confirmation parameter containing a public key

   Note that the COSE_Key structure may contain an "alg" or "key_ops"
   parameter. If such parameters are present, a client MUST NOT use
   a key that is not compatible with the profile or proof-of-
   possession algorithm according to those parameters.
COSE_Encrypted In this case the 'cnf' parameter contains an
   encrypted symmetric key destined for the client. The client is
   assumed to be able to decrypt the cihpertext of this parameter.
   The parameter is encoded as COSE_Encrypted object wrapping a
   COSE_Key object. Figure 8 shows an example of this type of
   encoding.

```
"cnf" : {
  "COSE_Encrypted" : {
     993(
       [ h'a1010a' # protected header : {"alg" : "AES-CCM-16-64-128"}
         "iv" : b64'ifUvZaHFgJM7UmGnjA',  # unprotected header
        b64'WXThuZo6TMCaZZqi6ef/8WHTjOdGk8kNzaIhIQ' # ciphertext
       ]
     )
  }
}
```

   Figure 8: Confirmation paramter containing an encrypted symmetric key

   The ciphertext here could e.g. contain a symmetric key as in
   Figure 9.

```
  {
    "kty" : "Symmetric",
    "kid" : b64'39Gqlw',
    "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
  }
```

Figure 9: Example plaintext of an encrypted cnf parameter

Key Identifier In this case the 'cnf' parameter references a key
    that is assumed to be previously known by the recipient. This
    allows clients that perform repeated requests for an access token
    for the same audience but e.g. with different scopes to omit key
    transport in the access token, token request and token response.
    Figure 10 shows such an example.

```
  "cnf" : {
    "kid" : b64'39Gqlw'
  }
```

Figure 10: A Confirmation parameter with just a key identifier

6.5. Mapping parameters to CBOR

All OAuth parameters in access token requests and responses are
mapped to CBOR types as follows and are given an integer key value to
save space.

```
/-------------------+----------+----------------\
| Parameter name    | CBOR Key | Major Type     |
|-------------------+----------+----------------|
| aud               | 3        | 3              |
| client_id         | 8        | 3 (text string)|
| client_secret     | 9        | 2 (byte string)|
| response_type     | 10       | 3              |
| redirect_uri      | 11       | 3              |
| scope             | 12       | 3              |
| state             | 13       | 3              |
| code              | 14       | 2              |
| error_description | 15       | 3              |
| error_uri         | 16       | 3              |
| grant_type        | 17       | 0 (unit)       |
| access_token      | 18       | 3              |
| token_type        | 19       | 0              |
| expires_in        | 20       | 0              |
| username          | 21       | 3              |
| password          | 22       | 3              |
| refresh_token     | 23       | 3              |
| cnf               | 24       | 5 (map)        |
| profile           | 25       | 3              |
\-------------------+----------+----------------/
```

Figure 11: CBOR mappings used in token requests

7. The 'Introspect' Endpoint

Token introspection [RFC7662] is 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 RFC 7662 [RFC7662]
for HTTP and JSON, this section defines adaptations to more
constrained environments using CoAP and CBOR.

Communication between the RS and the introspection endpoint at the AS
MUST be integrity protected and encrypted. Furthermore AS and RS
MUST perform mutual authentication. Finally the AS SHOULD verify
that the RS has the right to access introspection information about
the provided token. Profiles of this framework are expected to
specify how authentication and communication security is implemented.

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

7.1. RS-to-AS Request

   The RS sends a CoAP POST request to the introspection endpoint at the
   AS, the profile is expected to specify the Content-Type and wrapping
   of the payload. The payload MUST be encoded as a CBOR map with a
   'token' parameter containing the access token along with optional
   parameters representing additional context that is known by the RS to
   aid the AS in its response.

   The same parameters are required and optional as in section 2.1 of
   RFC 7662 [RFC7662].

   For example, Figure 12 shows a RS calling the token introspection
   endpoint at the AS to query about an OAuth 2.0 proof-of-possession
   token. Note that we assume a object security-based communication
   security profile for this example, therefore the Content-Type is
   "application/cose+cbor".

   Header: POST (Code=0.02)
   Uri-Host: "server.example.com"
   Uri-Path: "introspect"
   Content-Type: "application/cose+cbor"
   Payload:
   {
     "token" : b64'7gj0dXJQ43U',
     "token_type_hint" : "pop"
   }

                  Figure 12: Example introspection request.

7.2. AS-to-RS Response

   If the introspection request is authorized and successfully
   processed, the AS sends a response with the CoAP response 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 7.3.

   In a successful response, the AS encodes the response parameters in a
   CBOR map including with the same required and optional parameters as
   in section 2.2. of RFC 7662 [RFC7662] with the following additions:

   cnf
      OPTIONAL. This field contains information about the proof-of-
      possession key that binds the client to the access token. See
      Section 6.4.5 for more details on the formatting of the 'cnf'
      parameter.

   profile
      OPTIONAL. This indicates the profile that the RS MUST use with
      the client. See Section 6.4.4 for more details on the formatting
      of this parameter.

   client_token
      OPTIONAL. This parameter contains information that the RS MUST
      pass on to the client. See Section 7.4 for more details.

   For example, Figure 13 shows an AS response to the introspection
   request in Figure 12. Note that we assume a DTLS-based communication
   security profile for this example, therefore the Content-Type is
   "application/cbor".

   Header: Created Code=2.01)
   Content-Type: "application/cbor"
   Payload:
   {
     "active" : true,
     "scope" : "read",
     "profile" : "coap_dtls",
     "client_token" : b64'2QPhg0OhAQo ...
     (remainder of client token omitted for brevity)',
     "cnf" : {
       "COSE_Key" : {
         "kty" : "Symmetric",
         "kid" : b64'39Gqlw',
         "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
       }
     }
   }

                Figure 13: Example introspection response.

7.3. 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, the Content-Type MUST be set according to the
      specification of the communication security profile, and the
      content payload MUST be encoded as a CBOR map.
   o If the credentials used by the RS are invalid the AS MUST respond
      with the CoAP response code 4.01 (Unauthorized) and use the
      required and optional parameters from section 5.2 in RFC 6749
      [RFC6749].

   o If the RS does not have the right to perform this introspection
      request, the AS MUST respond with the CoAP response code 4.03
      (Forbidden). In this case no payload is returned.

   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".

7.4. Client Token

   EDITORIAL NOTE: We have tentatively introduced this concept and would
   specifically like feedback whether this is viewed as a useful
   addition to the framework.

   In cases where the client has limited connectivity and needs to get
   access to a previously unknown resource servers, this framework
   suggests the following approach: The client is pre-configured with a
   generic, long-term access token when it is commissioned. When the
   client then tries to access a RS it transmits this access token. The
   RS then performs token introspection to learn what access this token
   grants. In the introspection response, the AS also relays
   information for the client, such as the proof-of-possession key,
   through the RS. The RS passes on this Client Token to the client in
   response to the submission of the token.

   The client_token parameter is designed to carry such information, and
   is intended to be used as described in Figure 14.

```
                   Resource        Authorization
        Client      Server             Server
          |           |                  |
          |           |                  |
    C:   +--------------->|               |
          |   POST     |                  |
          |  Access Token |               |
          |           D:  +--------------->|
          |           | Introspection  |
          |           |     Request    |
          |           |                  |
          |           E:  +<--------------+
          |           | Introspection  |
          |           |    Response    |
          |           | + Client Token |
          |<--------------+               |
          |   2.01 Created |               |
          | + Client Token |
```
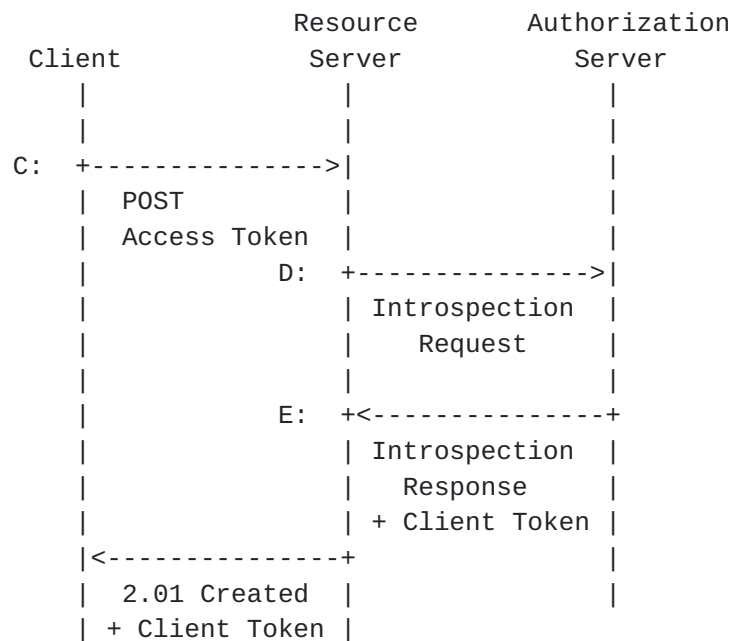
                Figure 14: Use of the client_token parameter.

   The client token is a COSE_Encrypted object, containing as payload a
   CBOR map with the following claims:

   cnf
      REQUIRED if the token type is 'pop', OPTIONAL otherwise. Contains
      information about the proof-of-possession key the client is to use
      with its access token. See Section 6.4.5.

   token_type
      OPTIONAL. See Section 6.4.3.

   profile
      REQUIRED. See Section 6.4.4.

   rs_cnf
      OPTIONAL. Contains information about the key that the RS uses to
      authenticate towards the client. If the key is symmetric then
      this claim MUST NOT be part of the Client Token, since this is the
      same key as the one specified through the 'cnf' claim. This claim
      uses the same encoding as the 'cnf' parameter. See Section 6.4.4.

   The AS encrypts this token using a key shared between the AS and the
   client, so that only the client can decrypt it and access its
   payload. How this key is established is out of scope of this
   framework.

7.5. **Mapping Introspection parameters to CBOR**

   The introspection request and response parameters are mapped to CBOR
   types as follows and are given an integer key value to save space.

```
         /-----------------+----------+-----------------\
         | Parameter name  | CBOR Key | Major Type      |
         |-----------------+----------+-----------------|
         | iss             | 1        | 3 (text string) |
         | sub             | 2        | 3               |
         | aud             | 3        | 3               |
         | exp             | 4        | 6 tag value 1   |
         | nbf             | 5        | 6 tag value 1   |
         | iat             | 6        | 6 tag value 1   |
         | cti             | 7        | 2 (byte string) |
         | client_id       | 8        | 3               |
         | scope           | 12       | 3               |
         | token_type      | 19       | 3               |
         | username        | 21       | 3               |
         | cnf             | 24       | 5 (map)         |
         | profile         | 25       | 0 (uint)        |
         | token           | 26       | 3               |
         | token_type_hint | 27       | 3               |
         | active          | 28       | 0               |
         | client_token    | 29       | 3               |
         | rs_cnf          | 30       | 5               |
         \-----------------+----------+-----------------/
```

          Figure 15: CBOR Mappings to Token Introspection Parameters.

8. **The Access Token**

   This framework RECOMMENDS the use of CBOR web token (CWT) as
   specified in [I-D.ietf-ace-cbor-web-token].

   In order to facilitate offline processing of access tokens, this
   draft specifies the "cnf" and "scope" claims for CBOR web tokens.

   The "scope" claim explicitly encodes the scope of a given access
   token. This claim follows the same encoding rules as defined in
   section 3.3 of [RFC6749]. The meaning of a specific scope value is
   application specific and expected to be known to the RS running that
   application.

   The "cnf" claim follows the same rules as specified for JSON web
   token in RFC7800 [RFC7800], except that it is encoded in CBOR in the
   same way as specified for the "cnf" parameter in Section 6.4.5.

8.1. The 'Authorization Information' Endpoint

   The access token, containing authorization information and
   information of the key 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 CoAP. Profiles of this framework MAY define other
   methods for token transport. Implementations conforming to this
   framework MUST implement this method of token transportation.

   The method consists of a /authz-info endpoint, implemented by the RS.
   A client using this method MUST make a POST request to /authz-info 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 2.04 (Changed).

   If the token is not valid, the RS MUST respond with the CoAP response
   code 4.01 (Unauthorized). If the token is valid but the audience of
   the token does not match the RS, the RS MUST respond with the CoAP
   response code 4.03 (Forbidden).

   The RS MAY make an introspection request to validate the token before
   responding to the POST /authz-info request. If the introspection
   response contains a client token (Section 7.4) then this token SHALL
   be included in the payload of the 2.04 (Changed) response.

   Profiles are expected to specify how the /authz-info endpoint is
   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.

8.2. Token Expiration

   Depending on the capabilities of the RS, there are various ways in
   which it can verify the validity of a received access token. We list
   the possibilities here including what functionality they require of
   the RS.

   o The token is a CWT/JWT and includes a '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 memo.
   o The RS verifies the validity of the token by performing an
      introspection request as specified in Section 7. 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 AS to
      RS).

   o The RS and the AS both store a sequence number linked to their
      common security association. The AS increments this number for
      each access token it issues and includes it in the access token,
      which is a CWT/JWT. The RS keeps track of the most recently
      received sequence number, and only accepts tokens as valid, that
      are in a certain range around this number. This method does only
      require the RS to keep track of the sequence number. The method
      does not provide timely expiration, but it makes sure that older
      tokens cease to be valid after a certain number of newer ones got
      issued. For a constrained RS with no network connectivity and no
      means of reliably measuring time, this is the best that can be
      achieved.

9. Security Considerations

   The entire document is about security. Security considerations
   applicable to authentication and authorization in RESTful
   environments provided in OAuth 2.0 [RFC6749] apply to this work, as
   well as the security considerations from [I-D.ietf-ace-actors].
   Furthermore [RFC6819] provides additional security considerations for
   OAuth which apply to IoT deployments as well.

   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. 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. If
   the access token contains the symmetric key, this symmetric key MUST
   be encrypted by the authorization server with a long-term key shared
   with the resource server.

   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. Using a single
   shared secret with multiple resource servers to simplify key
   management is NOT RECOMMENDED since the benefit from using the proof-
   of-possession concept is significantly reduced.

   Token replay is also not possible since an eavesdropper will also
   have to obtain the corresponding private key or shared secret that is
   bound to the access token. Nevertheless, it is good practice to
   limit the lifetime of the access token and therefore the lifetime of
   associated key.

The authorization server MUST offer confidentiality protection for
any interactions with the client. This step is extremely important
since the client will obtain the session 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 making the proof-of-possession security
model completely insecure. This framework relies on profiles to
define how confidentiality protection is provided, and additional
protection can be applied by encrypting the CWT as specified in
section 5.1 of [I-D.ietf-ace-cbor-web-token] to provide an additional
layer of protection for cases where keying material is conveyed, for
example, to a hardware security module.

Developers MUST ensure that the ephemeral credentials (i.e., the
private key or the session key) is 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.

Clients can at any time request a new proof-of-possession capable
access token. Using a refresh token to regularly request new access
tokens that are bound to fresh and unique keys is important if the
client has this capability. Keeping the lifetime of the access token
short allows the authorization server to use shorter 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 then
they SHOULD scope these access tokens to a specific permissions.

10. IANA Considerations

This specification registers new parameters for OAuth and establishes
registries for mappings to CBOR.

10.1. OAuth Introspection Response Parameter Registration

This specification registers the following parameters in the OAuth
introspection response parameters

o Name: "cnf"
o Description: Key to prove the right to use an access token, as
   defined in [RFC7800].
o Change Controller: IESG
o Specification Document(s): this document

o Name: "aud"
o Description: Reference to intended receiving RS, as defined in PoP
   token specification.
o Change Controller: IESG
o Specification Document(s): this document

o Name: "profile"
o Description: The communication and communication security profile
   used between client and RS, as defined in ACE profiles.
o Change Controller: IESG
o Specification Document(s): this document

o Name: "client_token"
o Description: Information that the RS MUST pass to the client e.g.
   about the proof-of-possession keys.
o Change Controller: IESG
o Specification Document(s): this document

o Name: "rs_cnf"
o Description: Describes the public key the RS uses to authenticate.
o Change Controller: IESG
o Specification Document(s): this document

10.2. OAuth Parameter Registration

This specification registers the following parameters in the OAuth
Parameters Registry

o Parameter name: "profile"
o Parameter usage location: token request, and token response
o Change Controller: IESG
o Specification Document(s): this document

o Name: "cnf"
o Description: Key to prove the right to use an access token, as
   defined in [RFC7800].
o Change Controller: IESG
o Specification Document(s): this document

10.3. OAuth Access Token Types

This specification registers the following new token type in the
OAuth Access Token Types Registry

o Name: "PoP"
o Description: A proof-of-possession token.
o Change Controller: IESG
o Specification Document(s): this document

10.4. Token Type Mappings

A new registry will be requested from IANA, entitled "Token Type
Mappings". The registry is to be created as Expert Review Required.

10.4.1. Registration Template

Token Type:
   Name of token type as registered in the OAuth token type registry
   e.g. "Bearer".
Mapped value:
   Integer representation for the token type value. The key value
   MUST be an integer in the range of 1 to 65536.
Change Controller:
   For Standards Track RFCs, list the "IESG". For others, give the
   name of the responsible party. Other details (e.g., postal
   address, email address, home page URI) may also be included.
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.

10.4.2. Initial Registry Contents

o Parameter name: "Bearer"
o Mapped value: 1
o Change Controller: IESG
o Specification Document(s): this document

o Parameter name: "pop"
o Mapped value: 2
o Change Controller: IESG
o Specification Document(s): this document

10.5. CBOR Web Token Claims

This specification registers the following new claims in the CBOR Web
Token (CWT) registry:

o Claim Name: "scope"
o Claim Description: The scope of an access token as defined in
   [RFC6749].
o Change Controller: IESG
o Specification Document(s): this document

o Claim Name: "cnf"

   o Claim Description: The proof-of-possession key of an access token
      as defined in [RFC7800].
   o Change Controller: IESG
   o Specification Document(s): this document

10.6. ACE Profile Registry

   A new registry will be requested from IANA, entitled "ACE Profile
   Registry". The registry is to be created as Expert Review Required.

10.6.1. Registration Template

   Profile name:
      Name of the profile to be included in the profile attribute.
   Profile description:
      Text giving an overview of the profile and the context it is
      developed for.
   Profile ID:
      Integer value to identify the profile. The value MUST be an
      integer in the range of 1 to 65536.
   Change Controller:
      For Standards Track RFCs, list the "IESG". For others, give the
      name of the responsible party. Other details (e.g., postal
      address, email address, home page URI) may also be included.
   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.

10.7. OAuth Parameter Mappings Registry

   A new registry will be requested from IANA, entitled "Token Endpoint
   CBOR Mappings Registry". The registry is to be created as Expert
   Review Required.

10.7.1. Registration Template

   Parameter name:
      OAuth Parameter name, refers to the name in the OAuth parameter
      registry e.g. "client_id".
   CBOR key value:
      Key value for the claim. The key value MUST be an integer in the
      range of 1 to 65536.
   Change Controller:
      For Standards Track RFCs, list the "IESG". For others, give the
      name of the responsible party. Other details (e.g., postal
      address, email address, home page URI) may also be included.

   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.

10.7.2. Initial Registry Contents

   o Parameter name: "aud"
   o CBOR key value: 3
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "client_id"
   o CBOR key value: 8
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "client_secret"
   o CBOR key value: 9
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "response_type"
   o CBOR key value: 10
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "redirect_uri"
   o CBOR key value: 11
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "scope"
   o CBOR key value: 12
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "state"
   o CBOR key value: 13
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "code"
   o CBOR key value: 14
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "error_description"
   o CBOR key value: 15
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "error_uri"
   o CBOR key value: 16
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "grant_type"
   o CBOR key value: 17
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "access_token"
   o CBOR key value: 18
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "token_type"
   o CBOR key value: 19
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "expires_in"
   o CBOR key value: 20
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "username"
   o CBOR key value: 21
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "password"
   o CBOR key value: 22
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "refresh_token"
   o CBOR key value: 23
   o Change Controller: IESG
   o Specification Document(s): this document

   o Parameter name: "cnf"
   o CBOR key value: 24
   o Change Controller: IESG

o Specification Document(s): this document

o Parameter name: "profile"
o CBOR key value: 25
o Change Controller: IESG
o Specification Document(s): this document

10.8. Introspection Endpoint CBOR Mappings Registry

A new registry will be requested from IANA, entitled "Introspection
Endpoint CBOR Mappings Registry". The registry is to be created as
Expert Review Required.

10.8.1. Registration Template

Response parameter name:
   Name of the response parameter as defined in the "OAuth Token
   Introspection Response" registry e.g. "active".
CBOR key value:
   Key value for the claim. The key value MUST be an integer in the
   range of 1 to 65536.
Change Controller:
   For Standards Track RFCs, list the "IESG". For others, give the
   name of the responsible party. Other details (e.g., postal
   address, email address, home page URI) may also be included.
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.

10.8.2. Initial Registry Contents

o Response parameter name: "iss"
o CBOR key value: 1
o Change Controller: IESG
o Specification Document(s): this document

o Response parameter name: "sub"
o CBOR key value: 2
o Change Controller: IESG
o Specification Document(s): this document

o Response parameter name: "aud"
o CBOR key value: 3
o Change Controller: IESG
o Specification Document(s): this document

      o Response parameter name: "exp"
      o CBOR key value: 4
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "nbf"
      o CBOR key value: 5
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "iat"
      o CBOR key value: 6
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "cti"
      o CBOR key value: 7
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "client_id"
      o CBOR key value: 8
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "scope"
      o CBOR key value: 12
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "token_type"
      o CBOR key value: 19
      o Change Controller: IESG
      o Specification Document(s): this document

      o Response parameter name: "username"
      o CBOR key value: 21
      o Change Controller: IESG
      o Specification Document(s): this document

      o Parameter name: "cnf"
      o CBOR key value: 24
      o Change Controller: IESG
      o Specification Document(s): this document

      o Parameter name: "profile"
      o CBOR key value: 25
      o Change Controller: IESG

   o Specification Document(s): this document

   o Response parameter name: "token"
   o CBOR key value: 26
   o Change Controller: IESG
   o Specification Document(s): this document

   o Response parameter name: "token_type_hint"
   o CBOR key value: 27
   o Change Controller: IESG
   o Specification Document(s): this document

   o Response parameter name: "active"
   o CBOR key value: 28
   o Change Controller: IESG
   o Specification Document(s): this document

   o Response parameter name: "client_token"
   o CBOR key value: 29
   o Change Controller: IESG
   o Specification Document(s): this document

   o Response parameter name: "rs_cnf"
   o CBOR key value: 30
   o Change Controller: IESG
   o Specification Document(s): this document

10.9. CoAP Option Number Registration

   This section registers the "Access-Token" CoAP Option Number in the
   "CoRE Parameters" sub-registry "CoAP Option Numbers" in the manner
   described in [RFC7252].

   Name

      Access-Token
   Number

      TBD
   Reference

      [This document].
   Meaning in Request

      Contains an Access Token according to [This document] containing
      access permissions of the client.
   Meaning in Response

      Not used in response
   Safe-to-Forward

      Yes
   Format

      Based on the observer the format is perceived differently. Opaque
      data to the client and CWT or reference token to the RS.
   Length

      Less then 255 bytes

11. Acknowledgments

   We would like to thank 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 ACRE proposal
   [I-D.seitz-ace-core-authz] which was one source of inspiration for
   this work. Finally, we would like to thank the ACE working group in
   general for their feedback.

   We would like to thank the authors of draft-ietf-oauth-pop-key-
   distribution, from where we copied large parts of our security
   considerations.

   Ludwig Seitz and Goeran Selander worked on this document as part of
   the CelticPlus project CyberWI, with funding from Vinnova.

12. References

12.1. Normative References

   [I-D.ietf-ace-cbor-web-token]
              Wahlstroem, E., Jones, M., Tschofenig, H., and S. Erdtman,
              "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-01
              (work in progress), July 2016.

   [I-D.ietf-cose-msg]
              Schaad, J., "CBOR Object Signing and Encryption (COSE)",
              draft-ietf-cose-msg-23 (work in progress), October 2016.

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

   [RFC6347]   Rescorla, E. and N. Modadugu, "Datagram Transport Layer
               Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,
               January 2012, <http://www.rfc-editor.org/info/rfc6347>.

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

   [RFC7662]   Richer, J., Ed., "OAuth 2.0 Token Introspection",
               RFC 7662, DOI 10.17487/RFC7662, October 2015,
               <http://www.rfc-editor.org/info/rfc7662>.

   [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,
               <http://www.rfc-editor.org/info/rfc7800>.

12.2. 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-04 (work in
               progress), September 2016.

   [I-D.ietf-oauth-device-flow]
               Denniss, W., Myrseth, S., Bradley, J., Jones, M., and H.
               Tschofenig, "OAuth 2.0 Device Flow", draft-ietf-oauth-
               device-flow-03 (work in progress), July 2016.

   [I-D.ietf-oauth-native-apps]
               Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps",
               draft-ietf-oauth-native-apps-05 (work in progress),
               October 2016.

   [I-D.seitz-ace-core-authz]
               Seitz, L., Selander, G., and M. Vucinic, "Authorization
               for Constrained RESTful Environments", draft-seitz-ace-
               core-authz-00 (work in progress), June 2015.

   [I-D.selander-ace-object-security]
               Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
               "Object Security of CoAP (OSCOAP)", draft-selander-ace-
               object-security-06 (work in progress), October 2016.

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

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246,
              DOI 10.17487/RFC5246, August 2008,
              <http://www.rfc-editor.org/info/rfc5246>.

   [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
              Format", RFC 6690, DOI 10.17487/RFC6690, August 2012,
              <http://www.rfc-editor.org/info/rfc6690>.

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

   [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,
              <http://www.rfc-editor.org/info/rfc6819>.

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

   [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
              Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March
              2014, <http://www.rfc-editor.org/info/rfc7159>.

   [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
              Constrained-Node Networks", RFC 7228,
              DOI 10.17487/RFC7228, May 2014,
              <http://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,
              <http://www.rfc-editor.org/info/rfc7231>.

   [RFC7519] 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>.

   [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, <http://www.rfc-editor.org/info/rfc7521>.

   [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,
              <http://www.rfc-editor.org/info/rfc7591>.

   [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,
              <http://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,
              <http://www.rfc-editor.org/info/rfc7959>.

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. 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
      so allows 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 to perform, 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 allows, 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
small amount of RAM and flash memory, which places limitations
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. Authentication and key establishment protocol,
e.g. the DTLS handshake, in comparison with assisted key
establishment also has an impact on memory and code.

User Interface Limitations:

Protecting access to resources is both an important security as
well as privacy feature. End users and enterprise customers do
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.
While we envision deployments to make use of CoAP we explicitly
want to support HTTP, HTTP/2 or specific protocols, such as
Bluetooth Smart communication, which does not necessarily use IP.

The latter raises the need for application layer security over the
various interfaces.

[Appendix B](#). Roles and Responsibilites

Resource Owner

*   Make sure that the RS is registered at the AS. This includes
    making known to the AS which profiles, token_types, 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 which is in charge
    of the RS.
*   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 RS and manage corresponding security contexts.
*   Register clients and including authentication credentials.
*   Allow Resource Owners to configure and update access control
    policies related to their registered RS'
*   Expose the /token endpoint to allow clients to request tokens.
*   Authenticate clients that wish to request a token.
*   Process a token request against the authorization policies
    configured for the RS.
*   Expose the /introspection endpoint that allows RS's to submit
    token introspection requests.
*   Authenticate RS's that wish to get an introspection response.
*   Process token introspection requests.
*   Optionally: Handle token revocation.

Client

   * Discover the AS in charge of the RS that is to be targeted with
     a request.
   * Submit the token request (A).

     + Authenticate towards 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 key (rpk) or certificate is used, make sure
       the AS has the right rpk or certificate for this client.
   * Process the access token and RS Information (B)

     + Check that the RS Information provides the necessary
       security parameters (e.g. PoP key, information on
       communication security protocols supported by the RS).
   * Send the token and request to the RS (C)

     + 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 (F) requirements of the Requesting
     Party, when issuing requests (e.g. minimum communication
     security requirements, trust anchors).
   * Register the client at the AS.

   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 the right AS.
     + Verify that the token applies to this RS.
     + Check that the token has not expired (if the token provides
       expiration information).
     + Check the token's integrity.
     + Store the token so that it can be retrieved in the context
       of a matching request.
   * 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.

Appendix C. Requirements on Profiles

   This section lists the requirements on profiles of this framework,
   for the convenience of a profile designer. All this information is
   also given in the appropriate sections of the main document, this is
   just meant as a checklist, to make it more easy to spot parts one
   might have missed.

   o Specify the discovery process of how the client finds the right AS
      for an RS it wants to send a request to.
   o Specify the communication protocol the client and RS the must use
      (e.g. CoAP).
   o Specify the security protocol the client and RS must use to
      protect their communication (e.g. OSCOAP or DTLS over CoAP).
      This must provide encryption and integrity protection.
   o Specify how the client and the RS mutually authenticate
   o Specify the Content-format of the protocol messages (e.g.
      "application/cbor" or "application/cose+cbor").
   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.
   o Specify a unique profile identifier.
   o Optionally specify how the RS talks to the AS for introspection.
   o Optionally specify how the client talks to the AS for requesting a
      token.
   o Specify how/if the /authz-info endpoint is protected.
   o Optionally define other methods of token transport than the
      /authz-info endpoint.

Appendix D. 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 a RS is

      performed. This requires the the security of the requests and
      responses between the clients and the RS to consider.

      Note: CBOR diagnostic notation is used for examples of requests and
      responses.

D.1. Local Token Validation

   In this scenario we consider the case where the resource server is
   offline, 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 16.

      A: The client first generates a public-private key pair used for
      communication security with the RS.
      The client sends the POST request to /token at the AS. The
      security of this request can be transport or application layer, it
      is up the the comunication security profile to define. In the
      example trasport layer identification of the AS is done and the
      client identifies with client_id and client_secret as in classic
      OAuth. 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 PoP token and RS Information. The PoP
      token contains the public key of the client, and the RS
      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', to protect the RS from replay attacks.
      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 we assume 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
        |          |    to identify the AS
        |          |
  A:   +-------->| Header: POST (Code=0.02)
        |  POST   | Uri-Path:"token"
        |          | Content-Type: application/cbor
        |          | Payload: <Request-Payload>
        |          |
  B:   |<--------+ Header: 2.05 Content
        |  2.05   | Content-Type: application/cbor
        |          | Payload: <Response-Payload>
        |          |
```
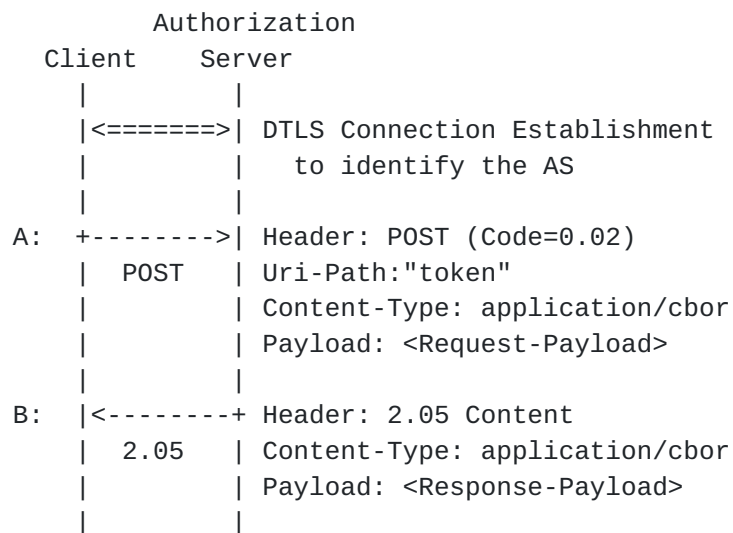
        Figure 16: Token Request and Response Using Client Credentials.

   The information contained in the Request-Payload and the Response-
   Payload is shown in Figure 17. Note that we assume a DTLS-based
   communication security profile for this example, therefore the
   Content-Type is "application/cbor".

```
   Request-Payload :
   {
     "grant_type" : "client_credentials",
     "aud" : "tempSensorInLivingRoom",
     "client_id" : "myclient",
     "client_secret" : "qwerty"
   }

   Response-Payload :
   {
     "access_token" : b64'SlAV32hkKG ...',
     "token_type" : "pop",
     "csp" : "DTLS",
     "cnf" : {
       "COSE_Key" : {
         "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
         "kty" : "EC",
         "crv" : "P-256",
         "x"   : b64'MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
         "y"   : b64'4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
       }
     }
   }
```

              Figure 17: Request and Response Payload Details.

The content of the access token is shown in Figure 18.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "jwk" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}
```

Figure 18: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 19 - Figure 20.

C: The client then sends the PoP token to the /authz-info endpoint
at the RS. This is a plain CoAP request, i.e. no transport or
application layer security between client and RS, since the token
is integrity protected between AS and RS. The RS verifies that
the PoP token was created by a known and trusted AS, is valid, and
responds to the client. The RS caches the security context
together with authorization information about this client
contained in the PoP token.

```
             Resource
  Client      Server
      |          |
C:   +-------->| Header: POST (Code=0.02)
      |  POST   | Uri-Path:"authz-info"
      |         | Payload: SlAV32hkKG ...
      |         |
      |<--------+ Header: 2.04 Changed
      |  2.04   |
      |         |
```

Figure 19: 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 the CoAP request GET to /temperature on RS over
       DTLS. The RS verifies that the request is authorized, based on
       previously established security context.
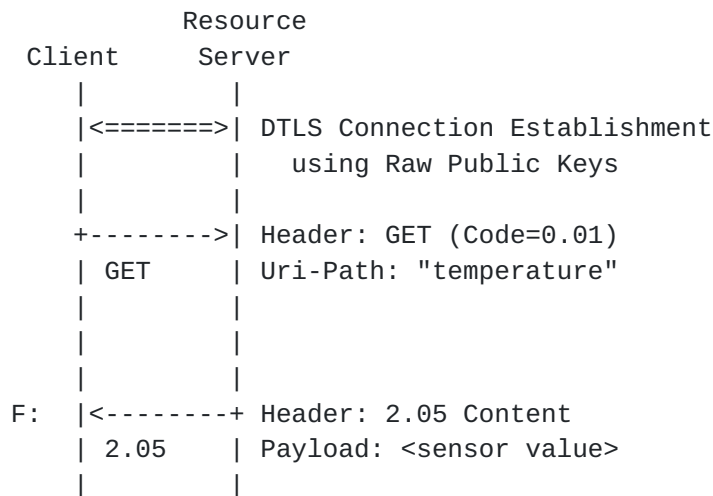   F: The RS responds with a resource representation over DTLS.

```
             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 20: Resource Request and Response protected by DTLS.

D.2. Introspection Aided Token Validation

   In this deployment scenario we assume that a client is not able to
   access the AS at the time of the access request. Since the RS is,
   however, connected to the back-end infrastructure it 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.
   The resource server may use its online connectivity to validate the
   access token with the authorization server, which is shown in the
   example below.

   In the example interactions between an offline client (key fob), a RS
   (online lock), and an AS is shown. We assume 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 21.

   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 is set quite wide to
start with and new values different form the original once can be
returned from introspection later on.

   A: The client sends the request using POST to /token at AS. The
   request contains the Audience parameter set to "PACS1337" (PACS,
   Physical Access System), a value the that the online door in
   question identifies itself with. The AS generates an access token
   as on opaque string, which it can match to the specific client, a
   targeted audience and a symmetric key. The security is provided
   by identifying the AS on transport layer using a pre shared
   security context (psk, rpk or certificate) and then the client is
   identified using client_id and client_secret as in classic OAuth
   B: The AS responds with the an access token and RS Information,
   the latter containing a symmetric key. Communication security
   between C and RS will be DTLS and PreSharedKey. The PoP key being
   used as the PreSharedKey.

```
          Authorization
   Client      Server
       |          |
       |          |
  A:   +-------->| Header: POST (Code=0.02)
       |   POST  | Uri-Path:"token"
       |         | Content-Type: application/cbor
       |         | Payload: <Request-Payload>
       |          |
  B:   |<--------+ Header: 2.05 Content
       |         | Content-Type: application/cbor
       |   2.05  | Payload: <Response-Payload>
       |          |
```

       Figure 21: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-
Payload is shown in Figure 22.

```
   Request-Payload:
   {
     "grant_type" : "client_credentials",
     "aud" : "lockOfDoor4711",
     "client_id" : "keyfob",
     "client_secret" : "qwerty"
   }

   Response-Payload:
   {
     "access_token" : b64'SlAV32hkKG ...'
     "token_type" : "pop",
     "csp" : "DTLS",
     "cnf" : {
       "COSE_Key" : {
         "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
         "kty" : "oct",
         "alg" : "HS256",
         "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
       }
     }
   }
```

            Figure 22: Request and Response Payload for C offline

   The access token in this case is just an opaque 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 forwards the token to the /introspect endpoint on the
      AS. Introspection assumes a secure connection between the AS and
      the RS, e.g. using transport of application layer security. In
      the example AS is identified using pre shared security context
      (psk, rpk or certificate) while RS is acting as client and is
      identified with client_id and client_secret.
      E: The AS provides the introspection response 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.
      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"
        |        | Content-Type: "application/cbor"
        |        | Payload: b64'SlAV32hkKG ...''
        |        |
        |        |         Authorization
        |        |            Server
        |        |     |
        |     D: +--------->| Header: POST (Code=0.02)
        |        |  POST    | Uri-Path: "introspect"
        |        |          | Content-Type: "application/cbor"
        |        |          | Payload: <Request-Payload>
        |        |          |
        |     E: |<---------+ Header: 2.05 Content
        |        |  2.05    | Content-Type: "application/cbor"
        |        |          | Payload: <Response-Payload>
        |        |          |
        |        |
        |<--------+ Header: 2.01 Created
        |  2.01  |
        |        |
```

               Figure 23: Token Introspection for C offline

     The information contained in the Request-Payload and the Response-
     Payload is shown in Figure 24.

```
   Request-Payload:
   {
     "token" : b64'SlAV32hkKG...',
     "client_id" : "FrontDoor",
     "client_secret" : "ytrewq"
   }

   Response-Payload:
   {
     "active" : true,
     "aud" : "lockOfDoor4711",
     "scope" : "open, close",
     "iat" : 1311280970,
     "cnf" : {
       "kid" : b64'JDLUhTMjU2IiwiY3R5Ijoi ...'
     }
   }
```

          Figure 24: 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 RS changing state of the door to
      locked.
      F: The RS responds with a appropriate over the secure DTLS
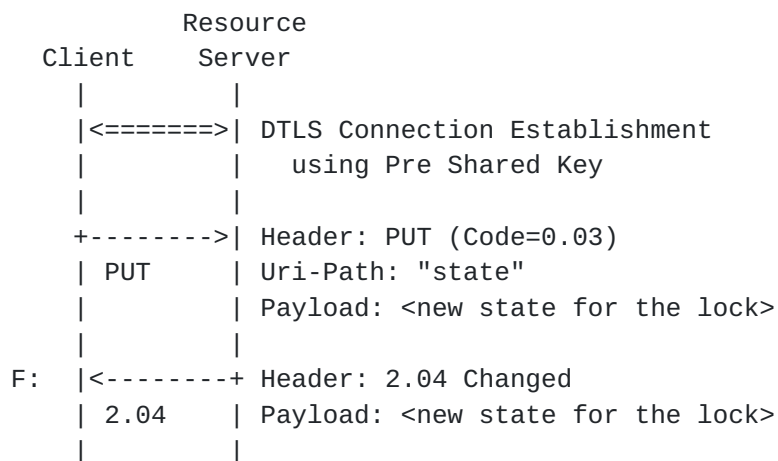   channel.

             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 25: Resource request and response protected by OSCOAP

Appendix E. Document Updates

E.1. 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.

E.2. 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, /introspect 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 posession.
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.

E.3. 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 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
    SICS
    Scheelevaegen 17
    Lund 223 70
    SWEDEN

    Email: ludwig@sics.se

    Goeran Selander
    Ericsson
    Faroegatan 6
    Kista 164 80
    SWEDEN

    Email: goran.selander@ericsson.com

    Erik Wahlstroem
    Sweden

    Email: erik@wahlstromtekniska.se

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

    Email: erdtman@spotify.com

    Hannes Tschofenig
    ARM Ltd.
    Hall in Tirol 6060
    Austria

    Email: Hannes.Tschofenig@arm.com