

ACE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 18, 2019

L. Seitz  
RISE  
G. Selander  
Ericsson  
E. Wahlstroem

S. Erdtman  
Spotify AB  
H. Tschofenig  
Arm Ltd.  
February 14, 2019

**Authentication and Authorization for Constrained Environments (ACE)  
using the OAuth 2.0 Framework (ACE-OAuth)  
draft-ietf-ace-oauth-authz-21**

**Abstract**

This specification defines a framework for authentication and authorization in Internet of Things (IoT) environments called ACE-OAuth. The framework is based on a set of building blocks including OAuth 2.0 and 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 <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on August 18, 2019.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Terminology</a>	<a href="#">5</a>
<a href="#">3.</a>	<a href="#">Overview</a>	<a href="#">6</a>
<a href="#">3.1.</a>	<a href="#">OAuth 2.0</a>	<a href="#">7</a>
<a href="#">3.2.</a>	<a href="#">CoAP</a>	<a href="#">10</a>
<a href="#">4.</a>	<a href="#">Protocol Interactions</a>	<a href="#">11</a>
<a href="#">5.</a>	<a href="#">Framework</a>	<a href="#">15</a>
<a href="#">5.1.</a>	<a href="#">Discovering Authorization Servers</a>	<a href="#">16</a>
<a href="#">5.1.1.</a>	<a href="#">Unauthorized Resource Request Message</a>	<a href="#">16</a>
<a href="#">5.1.2.</a>	<a href="#">AS Request Creation Hints</a>	<a href="#">17</a>
<a href="#">5.2.</a>	<a href="#">Authorization Grants</a>	<a href="#">19</a>
<a href="#">5.3.</a>	<a href="#">Client Credentials</a>	<a href="#">19</a>
<a href="#">5.4.</a>	<a href="#">AS Authentication</a>	<a href="#">20</a>
<a href="#">5.5.</a>	<a href="#">The Authorization Endpoint</a>	<a href="#">20</a>
<a href="#">5.6.</a>	<a href="#">The Token Endpoint</a>	<a href="#">20</a>
<a href="#">5.6.1.</a>	<a href="#">Client-to-AS Request</a>	<a href="#">21</a>
<a href="#">5.6.2.</a>	<a href="#">AS-to-Client Response</a>	<a href="#">24</a>
<a href="#">5.6.3.</a>	<a href="#">Error Response</a>	<a href="#">26</a>
<a href="#">5.6.4.</a>	<a href="#">Request and Response Parameters</a>	<a href="#">27</a>
<a href="#">5.6.4.1.</a>	<a href="#">Grant Type</a>	<a href="#">27</a>
<a href="#">5.6.4.2.</a>	<a href="#">Token Type</a>	<a href="#">28</a>
<a href="#">5.6.4.3.</a>	<a href="#">Profile</a>	<a href="#">28</a>
<a href="#">5.6.5.</a>	<a href="#">Mapping Parameters to CBOR</a>	<a href="#">29</a>
<a href="#">5.7.</a>	<a href="#">The Introspection Endpoint</a>	<a href="#">29</a>
<a href="#">5.7.1.</a>	<a href="#">Introspection Request</a>	<a href="#">30</a>
<a href="#">5.7.2.</a>	<a href="#">Introspection Response</a>	<a href="#">31</a>
<a href="#">5.7.3.</a>	<a href="#">Error Response</a>	<a href="#">32</a>
<a href="#">5.7.4.</a>	<a href="#">Mapping Introspection parameters to CBOR</a>	<a href="#">33</a>
<a href="#">5.8.</a>	<a href="#">The Access Token</a>	<a href="#">33</a>
<a href="#">5.8.1.</a>	<a href="#">The Authorization Information Endpoint</a>	<a href="#">34</a>
<a href="#">5.8.1.1.</a>	<a href="#">Verifying an Access Token</a>	<a href="#">35</a>



5.8.1.2. Protecting the Authorization Information Endpoint . . . . .	37
<a href="#">5.8.2. Client Requests to the RS . . . . .</a>	<a href="#">37</a>
<a href="#">5.8.3. Token Expiration . . . . .</a>	<a href="#">38</a>
<a href="#">5.8.4. Key Expriation . . . . .</a>	<a href="#">39</a>
<a href="#">6. Security Considerations . . . . .</a>	<a href="#">39</a>
<a href="#">6.1. Unprotected AS Request Creation Hints . . . . .</a>	<a href="#">41</a>
6.2. Minimal security requirements for communication . . . . .	41
<a href="#">6.3. Use of Nonces for Replay Protection . . . . .</a>	<a href="#">42</a>
<a href="#">6.4. Combining profiles . . . . .</a>	<a href="#">42</a>
<a href="#">6.5. Unprotected Information . . . . .</a>	<a href="#">42</a>
<a href="#">6.6. Identifying audiences . . . . .</a>	<a href="#">43</a>
<a href="#">6.7. Denial of service against or with Introspection . . . . .</a>	<a href="#">44</a>
<a href="#">7. Privacy Considerations . . . . .</a>	<a href="#">44</a>
<a href="#">8. IANA Considerations . . . . .</a>	<a href="#">45</a>
<a href="#">8.1. ACE Authorization Server Request Creation Hints . . . . .</a>	<a href="#">45</a>
<a href="#">8.2. OAuth Extensions Error Registration . . . . .</a>	<a href="#">46</a>
<a href="#">8.3. OAuth Error Code CBOR Mappings Registry . . . . .</a>	<a href="#">46</a>
<a href="#">8.4. OAuth Grant Type CBOR Mappings . . . . .</a>	<a href="#">46</a>
<a href="#">8.5. OAuth Access Token Types . . . . .</a>	<a href="#">47</a>
<a href="#">8.6. OAuth Access Token Type CBOR Mappings . . . . .</a>	<a href="#">47</a>
<a href="#">8.6.1. Initial Registry Contents . . . . .</a>	<a href="#">48</a>
<a href="#">8.7. ACE Profile Registry . . . . .</a>	<a href="#">48</a>
<a href="#">8.8. OAuth Parameter Registration . . . . .</a>	<a href="#">49</a>
<a href="#">8.9. OAuth Parameters CBOR Mappings Registry . . . . .</a>	<a href="#">49</a>
<a href="#">8.10. OAuth Introspection Response Parameter Registration . . . . .</a>	<a href="#">49</a>
8.11. OAuth Token Introspection Response CBOR Mappings Registry . . . . .	50
<a href="#">8.12. JSON Web Token Claims . . . . .</a>	<a href="#">50</a>
<a href="#">8.13. CBOR Web Token Claims . . . . .</a>	<a href="#">51</a>
<a href="#">8.14. Media Type Registrations . . . . .</a>	<a href="#">52</a>
<a href="#">8.15. CoAP Content-Format Registry . . . . .</a>	<a href="#">53</a>
<a href="#">8.16. Expert Review Instructions . . . . .</a>	<a href="#">53</a>
<a href="#">9. Acknowledgments . . . . .</a>	<a href="#">54</a>
<a href="#">10. References . . . . .</a>	<a href="#">54</a>
<a href="#">10.1. Normative References . . . . .</a>	<a href="#">54</a>
<a href="#">10.2. Informative References . . . . .</a>	<a href="#">56</a>
<a href="#">Appendix A. Design Justification . . . . .</a>	<a href="#">59</a>
<a href="#">Appendix B. Roles and Responsibilities . . . . .</a>	<a href="#">62</a>
<a href="#">Appendix C. Requirements on Profiles . . . . .</a>	<a href="#">64</a>
<a href="#">Appendix D. Assumptions on AS knowledge about C and RS . . . . .</a>	<a href="#">65</a>
<a href="#">Appendix E. Deployment Examples . . . . .</a>	<a href="#">66</a>
<a href="#">E.1. Local Token Validation . . . . .</a>	<a href="#">66</a>
<a href="#">E.2. Introspection Aided Token Validation . . . . .</a>	<a href="#">70</a>
<a href="#">Appendix F. Document Updates . . . . .</a>	<a href="#">74</a>
<a href="#">F.1. Verion -20 to 21 . . . . .</a>	<a href="#">74</a>
<a href="#">F.2. Verion -19 to 20 . . . . .</a>	<a href="#">74</a>
<a href="#">F.3. Version -18 to -19 . . . . .</a>	<a href="#">74</a>
<a href="#">F.4. Version -17 to -18 . . . . .</a>	<a href="#">75</a>



<a href="#">F.5.</a>	Version -16 to -17	<a href="#">75</a>
<a href="#">F.6.</a>	Version -15 to -16	<a href="#">75</a>
<a href="#">F.7.</a>	Version -14 to -15	<a href="#">75</a>
<a href="#">F.8.</a>	Version -13 to -14	<a href="#">76</a>
<a href="#">F.9.</a>	Version -12 to -13	<a href="#">76</a>
<a href="#">F.10.</a>	Version -11 to -12	<a href="#">76</a>
<a href="#">F.11.</a>	Version -10 to -11	<a href="#">76</a>
<a href="#">F.12.</a>	Version -09 to -10	<a href="#">76</a>
<a href="#">F.13.</a>	Version -08 to -09	<a href="#">77</a>
<a href="#">F.14.</a>	Version -07 to -08	<a href="#">77</a>
<a href="#">F.15.</a>	Version -06 to -07	<a href="#">77</a>
<a href="#">F.16.</a>	Version -05 to -06	<a href="#">78</a>
<a href="#">F.17.</a>	Version -04 to -05	<a href="#">78</a>
<a href="#">F.18.</a>	Version -03 to -04	<a href="#">78</a>
<a href="#">F.19.</a>	Version -02 to -03	<a href="#">78</a>
<a href="#">F.20.</a>	Version -01 to -02	<a href="#">79</a>
<a href="#">F.21.</a>	Version -00 to -01	<a href="#">79</a>
Authors' Addresses		<a href="#">80</a>

## **[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 can be a complex task.

While prior work on authorization solutions for the Web and for the mobile environment also applies to the Internet of Things (IoT) environment, many IoT devices are constrained, for example, in terms of processing capabilities, available memory, etc. For web applications on constrained nodes, this specification RECOMMENDS the use of CoAP [[RFC7252](#)] as replacement for HTTP.

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. Requirements on profiles are described at contextually appropriate places throughout this specification, and also summarized in [Appendix C](#).

## 2. Terminology

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

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

Since exchanges in this specification are described as RESTful protocol interactions, HTTP [[RFC7231](#)] offers useful terminology.

Terminology for entities in the architecture is defined in OAuth 2.0 [[RFC6749](#)] such as client (C), resource server (RS), and authorization server (AS).

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as token and introspection at the AS and authz-info at the RS (see [Section 5.8.1](#) for a definition of the authz-info endpoint). The CoAP [[RFC7252](#)] definition, which is "An entity participating in the CoAP protocol" is not used in this specification.

The specifications in this document is called the "framework" or "ACE framework". When referring to "profiles of this framework" it refers to additional specifications that define the use of this specification with concrete transport, and communication security protocols (e.g., CoAP over DTLS).





We use the term "Access Information" for parameters other than the access token provided to the client by the AS to enable it to access the RS (e.g. public key of the RS, profile supported by RS).

We use the term "Authorization Information" to denote all information, including the claims of relevant access tokens, that an RS uses to determine whether an access request should be granted.

### 3. Overview

This specification defines the ACE framework for authorization in the Internet of Things environment. It consists of a set of building blocks.

The basic block is the OAuth 2.0 [\[RFC6749\]](#) framework, which enjoys widespread deployment. Many IoT devices can support OAuth 2.0 without any additional extensions, but for certain constrained settings additional profiling is needed.

Another building block is the lightweight web transfer protocol CoAP [\[RFC7252\]](#), for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP, which further reduces overhead and message exchanges. While this specification defines extensions for the use of OAuth over CoAP, other underlying protocols are not prohibited from being supported in the future, such as HTTP/2, MQTT, BLE and QUIC.

A third building block is CBOR [\[RFC7049\]](#), for encodings where JSON [\[RFC8259\]](#) is not sufficiently compact. CBOR is a binary encoding designed for small code and message size, which may be used for encoding of self contained tokens, and also for encoding payload transferred in protocol messages.

A fourth building block is the compact CBOR-based secure message format COSE [\[RFC8152\]](#), which enables application layer security as an alternative or complement to transport layer security (DTLS [\[RFC6347\]](#) or TLS [\[RFC8446\]](#)). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which is an extension to the OAuth tokens. The default token format is defined in CBOR web token (CWT) [\[RFC8392\]](#). Application layer security for CoAP using COSE can be provided with OSCORE [\[I-D.ietf-core-object-security\]](#).

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. It is important to cover the wide range of possible interworking use cases and the different requirements from a security point of view. Once IoT deployments mature, popular deployment variants will be documented in the form of ACE profiles.

### [3.1.](#) **OAuth 2.0**

The OAuth 2.0 authorization framework enables a client to obtain scoped access to a resource with the permission of a resource owner. Authorization information, or references to it, is passed between the nodes using access tokens. These access tokens are issued to clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

A number of OAuth 2.0 terms are used within this specification:

The token and introspection Endpoints:

The AS hosts the token endpoint that allows a client to request access tokens. The client makes a POST request to the token endpoint on the AS and receives the access token in the response (if the request was successful).

In some deployments, a token introspection endpoint is provided by the AS, which can be used by the RS if it needs to request additional information regarding a received access token. The RS makes a POST request to the introspection endpoint on the AS and receives information about the access token in the response. (See "Introspection" below.)

Access Tokens:

Access tokens are credentials needed to access protected resources. An access token is a data structure representing authorization permissions issued by the AS to the client. Access tokens are generated by the AS and consumed by the RS. The access token content is opaque to the client.

Access tokens can have different formats, and various methods of utilization (e.g., cryptographic properties) based on the security requirements of the given deployment.



#### Refresh Tokens:

Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token (i.e., step (B) in Figure 1).

A refresh token in OAuth 2.0 is a string representing the authorization granted to the client by the resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers. In this framework, refresh tokens are encoded in binary instead of strings, if used.

#### Proof of Possession Tokens:

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 access tokens (or PoP access 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 access token unless specifically stated otherwise.

The key bound to the access token (the PoP key) may use either symmetric or asymmetric cryptography. The appropriate choice of the kind of cryptography depends on the constraints of the IoT devices as well as on the security requirements of the use case.

#### Symmetric PoP key:

The AS generates a random symmetric PoP key. The key is either stored to be returned on introspection calls or encrypted and



included in the 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 [RFC8392]) protected by a cryptographic wrapper (e.g., COSE [RFC8152]). The choice of PoP key does not necessarily imply a specific credential type for the integrity protection of the token.

#### Scopes and Permissions:

In OAuth 2.0, the client specifies the type of permissions it is seeking to obtain (via the scope parameter) in the access token request. In turn, the AS may use the scope response parameter to inform the client of the scope of the access token issued. As the client could be a constrained device as well, this specification defines the use of CBOR encoding as data format, see [Section 5](#), to request scopes and to be informed what scopes the access token actually authorizes.

The values of the scope parameter in OAuth 2.0 are expressed as a list of space-delimited, case-sensitive strings, with a semantic that is well-known to the AS and the RS. More details about the concept of scopes is found under [Section 3.3 in \[RFC6749\]](#).

#### Claims:

Information carried in the access token or returned from introspection, called claims, is in the form of name-value pairs. An access token may, for example, include a claim identifying the AS that issued the token (via the "iss" claim) and what audience the access token is intended for (via the "aud" claim). The





audience of an access token can be a specific resource or one or many resource servers. The resource owner policies influence what claims are put into the access token by the authorization server.

While the structure and encoding of the access token varies throughout deployments, a standardized format has been defined with the JSON Web Token (JWT) [[RFC7519](#)] where claims are encoded as a JSON object. In [[RFC8392](#)], an equivalent format using CBOR encoding (CWT) has been defined.

#### Introspection:

Introspection is a method for a resource server to query the authorization server for the active state and content of a received access token. This is particularly useful in those cases where the authorization decisions are very dynamic and/or where the received access token itself is an opaque reference rather than a self-contained token. More information about introspection in OAuth 2.0 can be found in [[RFC7662](#)].

### **[3.2.](#) CoAP**

CoAP is an application layer protocol similar to HTTP, but specifically designed for constrained environments. CoAP typically uses datagram-oriented transport, such as UDP, where reordering and loss of packets can occur. A security solution needs to take the latter aspects into account.

While HTTP uses headers and query strings to convey additional information about a request, CoAP encodes such information into header parameters called 'options'.

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [[RFC7959](#)]. However, blockwise transfer does not increase the size limits of CoAP options, therefore data encoded in options has to be kept small.

Transport layer security for CoAP can be provided by DTLS or TLS [[RFC6347](#)][[RFC8446](#)] [[I-D.ietf-tls-dtls13](#)]. CoAP defines a number of proxy operations that require transport layer security to be terminated at the proxy. One approach for protecting CoAP communication end-to-end through proxies, and also to support security for CoAP over a different transport in a uniform way, is to provide security at the application layer using an object-based security mechanism such as COSE [[RFC8152](#)].



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

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

#### 4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the token endpoint and optionally the introspection endpoint. A client obtains an access token, and optionally a refresh token, from an AS using the token endpoint and subsequently presents the access token to a RS to gain access to a protected resource. In most deployments the RS can process the access token locally, however in some cases the RS may present it to the AS via the introspection endpoint to get fresh information. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in [Section 3.1](#).

The OAuth 2.0 framework defines a number of "protocol flows" via grant types, which have been extended further with extensions to OAuth 2.0 (such as [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 \[RFC7521\]](#)) and the Client Credentials Grant (described in [Section 4.4 of \[RFC7521\]](#)). The Authorization Code Grant is a good fit for use with apps running on smart phones and tablets that request access to IoT devices, a common scenario in the smart home environment, where users need to go through an authentication and authorization phase (at least during the initial setup phase). The native apps guidelines described in [[RFC8252](#)] are applicable to this use case. The Client Credential Grant is a good fit for use with IoT devices where the OAuth client itself is constrained. In such a case, the resource owner has pre-arranged access rights for the client with the authorization server, which is often accomplished using a commissioning tool.

The consent of the resource owner, for giving a client access to a protected resource, can be provided dynamically as in the traditional OAuth flows, or it could be pre-configured by the resource owner as authorization policies at the AS, which the AS evaluates when a token request arrives. The resource owner and the requesting party (i.e., client owner) are not shown in Figure 1.



This framework supports a wide variety of communication security mechanisms between the ACE entities, such as client, AS, and RS. It is assumed that the client has been registered (also called enrolled or onboarded) to an AS using a mechanism defined outside the scope of this document. In practice, various techniques for onboarding have been used, such as factory-based provisioning or the use of commissioning tools. Regardless of the onboarding technique, this provisioning procedure implies that the client and the AS exchange credentials and configuration parameters. These credentials are used to mutually authenticate each other and to protect messages exchanged between the client and the AS.

It is also assumed that the RS has been registered with the AS, potentially in a similar way as the client has been registered with the AS. Established keying material between the AS and the RS allows the AS to apply cryptographic protection to the access token to ensure that its content cannot be modified, and if needed, that the content is confidentiality protected.

The keying material necessary for establishing communication security between C and RS is dynamically established as part of the protocol described in this document.

At the start of the protocol, there is an optional discovery step where the client discovers the resource server and the resources this server hosts. In this step, the client might also determine what permissions are needed to access the protected resource. A generic procedure is described in [Section 5.1](#), profiles MAY define other procedures for discovery.

In Bluetooth Low Energy, for example, advertisements are broadcasted by a peripheral, including information about the primary services. In CoAP, as a second example, a client can make a request to `"/.well-known/core"` to obtain information about available resources, which are returned in a standardized format as described in [[RFC6690](#)].



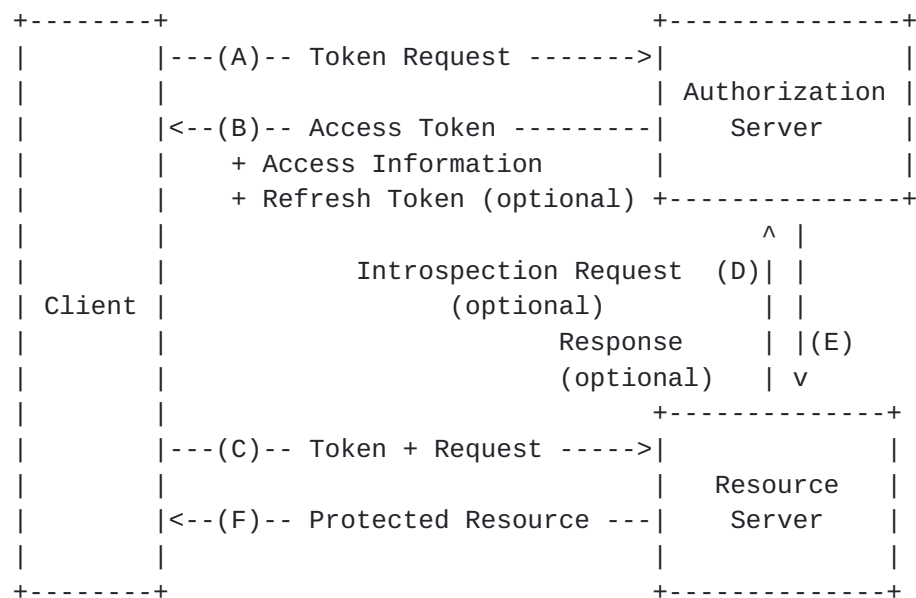


Figure 1: Basic Protocol Flow.

**Requesting an Access Token (A):**

The client makes an access token request to the token endpoint at the AS. This framework assumes the use of PoP access tokens (see [Section 3.1](#) for a short description) wherein the AS binds a key to an access token. The client may include permissions it seeks to obtain, and information about the credentials it wants to use (e.g., symmetric/asymmetric cryptography or a reference to a specific credential).

**Access Token Response (B):**

If the AS successfully processes the request from the client, it returns an access token and optionally a refresh token (note that only certain grant types support refresh tokens). It can also return additional parameters, referred to as "Access Information". In addition to the response parameters defined by OAuth 2.0 and the PoP access token extension, this framework defines parameters that can be used to inform the client about capabilities of the RS. More information about these parameters can be found in [Section 5.6.4](#).

**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 Access Information obtained from the AS.

The Client and the RS mutually authenticate using the security protocol specified in the profile (see step B) and the keys obtained in the access token or the Access Information. The RS verifies that the token is integrity protected 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.

#### Token Introspection Request (D):

A resource server may be configured to introspect the access token by including it in a request to the introspection endpoint at that AS. Token introspection over CoAP is defined in [Section 5.7](#) 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.

#### 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, it cannot be assumed that the client and RS are part of a common key infrastructure, so the AS provisions credentials or associated information to allow mutual authentication. 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 [[I-D.ietf-ace-cwt-proof-of-possession](#)] indicating what key is used for proof-of-possession. If a client needs to submit a new access token, e.g., to obtain additional access rights, they can request that the AS binds this token to the same key as the previous one.

### ACE Profiles

The client or RS may be limited in the encodings or protocols it supports. To support a variety of different deployment settings, specific interactions between client and RS are defined in an ACE profile. In ACE framework the AS is expected to manage the matching of compatible profile choices between a client and an RS. The AS informs the client of the selected profile using the "profile" parameter in the token response.

OAuth 2.0 requires the use of TLS both to protect the communication between AS and client when requesting an access token; between client and RS when accessing a resource and between AS and RS if introspection is used. In constrained settings TLS is not always feasible, or desirable. Nevertheless it is REQUIRED that the data exchanged with the AS is encrypted, integrity protected and protected against message replay. It is also REQUIRED that the AS and the endpoint communicating with it (client or RS) perform mutual



authentication. Furthermore it MUST be assured that responses are bound to the requests in the sense that the receiver of a response can be certain that the response actually belongs to a certain request.

Profiles MUST specify a communication security protocol that provides the features required above.

In OAuth 2.0 the communication with the Token and the Introspection endpoints at the AS is assumed to be via HTTP and may use Uri-query parameters. When profiles of this framework use CoAP instead, this framework REQUIRES 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. Profiles that use CBOR encoding of protocol message parameters MUST use the media format 'application/ace+cbor', unless the protocol message is wrapped in another Content-Format (e.g. object security). If CoAP is used for communication, the Content-Format MUST be abbreviated with the ID: 19 (see [Section 8.15](#)).

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. If CoAP is used, this framework REQUIRES the use of CBOR [[RFC7049](#)] instead of JSON. Depending on the profile, the CBOR payload MAY be enclosed in a non-CBOR cryptographic wrapper.

## **[5.1](#). Discovering Authorization Servers**

In order to determine the AS in charge of a resource hosted at the RS, C MAY send an initial Unauthorized Resource Request message to RS. RS then denies the request and sends the address of its AS back to C.

Instead of the initial Unauthorized Resource Request message, other discovery methods may be used, or the client may be pre-provisioned with the address of the AS.

### **[5.1.1](#). Unauthorized Resource Request Message**

The optional Unauthorized Resource Request message is a request for a resource hosted by RS for which no proper authorization is granted. RS MUST treat any request for a protected resource as Unauthorized Resource Request message when any of the following holds:

- o The request has been received on an unprotected channel.



- o RS has no valid access token for the sender of the request regarding the requested action on that resource.
- o RS has a valid access token for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The authz-info endpoint **MUST NOT** be protected as specified above, in order to allow clients to upload access tokens to RS (cf. [Section 5.8.1](#)).

Unauthorized Resource Request messages **MUST** be denied with a client error response. In this response, the Resource Server **SHOULD** provide proper AS Request Creation Hints to enable the Client to request an access token from RS's AS as described in [Section 5.1.2](#).

The handling of all client requests (including unauthorized ones) by the RS is described in [Section 5.8.2](#).

#### **[5.1.2](#). AS Request Creation Hints**

The AS Request Creation Hints message is sent by RS as a response to an Unauthorized Resource Request message (see [Section 5.1.1](#)) to help the sender of the Unauthorized Resource Request message in acquiring a valid access token. The AS Request Creation Hints message is CBOR map, with a **MANDATORY** element "AS" specifying an absolute URI (see [Section 4.3 of \[RFC3986\]](#)) that identifies the AS in charge of RS.

The message can also contain the following **OPTIONAL** parameters:

- o A "audience" element containing a suggested audience that the client should request towards the AS.
- o A "kid" element containing the key identifier of a key used in an existing security association between the client and the RS. The RS expects the client to request an access token bound to this key, in order to avoid having to re-establish the security association.
- o A "nonce" element containing a nonce generated by RS to ensure freshness in case that the RS and AS do not have synchronized clocks.
- o A "scope" element containing the suggested scope that the client should request towards the AS.





Figure 2 summarizes the parameters that may be part of the AS Request Creation Hints.

/-----+-----+-----\		
Name	CBOR Key	Value Type
-----+-----+-----		
AS	0	text string
kid	4	byte string
nonce	5	byte string
scope	9	text or byte string
audience	18	text string
\-----+-----+-----/		

Figure 2: AS Request Creation Hints

Note that the schema part of the AS parameter may need to be adapted to the security protocol that is used between the client and the AS. Thus the example AS value "coap://as.example.com/token" might need to be transformed to "coaps://as.example.com/token". It is assumed that the client can determine the correct schema part on its own depending on the way it communicates with the AS.

Figure 3 shows an example for an AS Request Creation Hints message payload using CBOR [\[RFC7049\]](#) diagnostic notation, using the parameter names instead of the CBOR keys for better human readability.

```
4.01 Unauthorized
Content-Format: application/ace+cbor
{AS: "coaps://as.example.com/token",
 audience: "coaps://rs.example.com",
 nonce: h'e0a156bb3f',
 scope: "rTempC"
}
```

Figure 3: AS Request Creation Hints payload example

In this example, the attribute AS points the receiver of this message to the URI "coaps://as.example.com/token" to request access permissions. The originator of the AS Request Creation Hints payload (i.e., RS) uses a local clock that is loosely synchronized with a time scale common between RS and AS (e.g., wall clock time). Therefore, it has included a parameter "nonce" for replay attack prevention.

Figure 4 illustrates the mandatory to use binary encoding of the message payload shown in Figure 3.



```
a2          # map(2)
  00        # unsigned(0) (=AS)
  78 1c     # text(28)
    636f6170733a2f2f61732e657861
    6d706c652e636f6d2f746f6b656e # "coaps://as.example.com/token"
  05        # unsigned(5) (=nonce)
  45        # bytes(5)
    e0a156bb3f
```

Figure 4: AS Request Creation Hints example encoded in CBOR

## 5.2. Authorization Grants

To request an access token, the client obtains authorization from the resource owner or uses its client credentials as grant. The authorization is expressed in the form of an authorization grant.

The OAuth framework [[RFC6749](#)] defines four grant types. The grant types can be split up into two groups, those granted on behalf of the resource owner (password, authorization code, implicit) and those for the client (client credentials). Further grant types have been added later, such as [[RFC7521](#)] defining an assertion-based authorization grant.

The grant type is selected depending on the use case. In cases where the client acts on behalf of the resource owner, authorization code grant is recommended. If the client acts on behalf of the resource owner, but does not have any display or very limited interaction possibilities it is recommended to use the device code grant defined in [[I-D.ietf-oauth-device-flow](#)]. In cases where the client does not act on behalf of the resource owner, client credentials grant is recommended.

For details on the different grant types, see the OAuth 2.0 framework [[RFC6749](#)]. The OAuth 2.0 framework provides an extension mechanism for defining additional grant types so profiles of this framework MAY define additional grant types, if needed.

## 5.3. Client Credentials

Authentication of the client is mandatory independent of the grant type when requesting the access token from the token endpoint. In the case of client credentials grant type, the authentication and grant coincide.

Client registration and provisioning of client credentials to the client is out of scope for this specification.



The OAuth framework [[RFC6749](#)] defines one client credential type, client id and client secret. [[I-D.erdman-ace-rpcc](#)] adds raw-public-key and pre-shared-key to the client credentials types. Profiles of this framework MAY extend with additional client credentials client certificates.

#### **[5.4.](#) AS Authentication**

Client credential does not, by default, authenticate the AS that the client connects to. In classic OAuth, the AS is authenticated with a TLS server certificate.

Profiles of this framework MUST specify how clients authenticate the AS and how communication security is implemented, otherwise server side TLS certificates, as defined by OAuth 2.0, are required.

#### **[5.5.](#) The Authorization Endpoint**

The authorization endpoint is used to interact with the resource owner and obtain an authorization grant in certain grant flows. Since it requires the use of a user agent (i.e., browser), it is not expected that these types of grant flow will be used by constrained clients. This endpoint is therefore out of scope for this specification. Implementations should use the definition and recommendations of [[RFC6749](#)] and [[RFC6819](#)].

If clients involved cannot support HTTP and TLS, profiles MAY define mappings for the authorization endpoint.

#### **[5.6.](#) The Token Endpoint**

In standard OAuth 2.0, the AS provides the token endpoint for submitting access token requests. This framework extends the functionality of the token endpoint, giving the AS the possibility to help the client and RS to establish shared keys or to exchange their public keys. Furthermore, this framework defines encodings using CBOR, as a substitute for JSON.

The endpoint may, however, be exposed over HTTPS as in classical OAuth or even other transports. A profile MUST define the details of the mapping between the fields described below, and these transports. If HTTPS is used, JSON or CBOR payloads may be supported. If JSON payloads are used, the semantics of [Section 4](#) of the OAuth 2.0 specification MUST be followed (with additions as described below). If CBOR payload is supported, the semantics described below MUST be followed.



For the AS to be able to issue a token, the client MUST be authenticated and present a valid grant for the scopes requested. Profiles of this framework MUST specify how the AS authenticates the client and how the communication between client and AS is protected.

The default name of this endpoint in an url-path is '/token', however implementations are not required to use this name and can define their own instead.

The figures of this section use CBOR diagnostic notation without the integer abbreviations for the parameters or their values for illustrative purposes. Note that implementations MUST use the integer abbreviations and the binary CBOR encoding, if the CBOR encoding is used.

#### **5.6.1. Client-to-AS Request**

The client sends a POST request to the token endpoint at the AS. The profile MUST specify how the communication is protected. The content of the request consists of the parameters specified in [Section 4](#) of the OAuth 2.0 specification [[RFC6749](#)] with the exception of the "grant\_type" parameter, which is OPTIONAL in the context of this framework (as opposed to REQUIRED in [RFC6749](#)). If that parameter is missing, the default value "client\_credentials" is implied.

Furthermore the "audience" parameter from [[I-D.ietf-oauth-token-exchange](#)] can be used to request an access token bound to a specific audience.

In addition to these parameters, a client MUST be able to use the parameters from [[I-D.ietf-ace-oauth-params](#)] in an access token request to the token endpoint and the AS MUST be able to process these additional parameters.

If CBOR is used then this parameter MUST be encoded as a CBOR map. The "scope" parameter can be formatted as specified in [[RFC6749](#)] and additionally as a byte string, in order to allow compact encoding of complex scopes.

When HTTP is used as a transport then the client makes a request to the token endpoint by sending the parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body, as defined in [RFC 6749](#).

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





Figure 5 shows a request for a token with a symmetric proof-of-possession key. The content is displayed in CBOR diagnostic notation, without abbreviations for better readability.

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

Figure 5: Example request for an access token bound to a symmetric key.

Figure 6 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example OSCORE [\[I-D.ietf-core-object-security\]](#) is used to provide object-security, therefore the Content-Format is "application/oscore" wrapping the "application/ace+cbor" type content. Also note that in this example the audience is implicitly known by both client and AS. Furthermore note that this example uses the "req\_cnf" parameter from [\[I-D.ietf-ace-oauth-params\]](#).



```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
OSCORE: 0x19, 0x05, 0x05, 0x44, 0x61, 0x6c, 0x65, 0x6b
Content-Format: "application/oscore"
Payload:
  0x44025d1 ... (full payload omitted for brevity) ... 68b3825e
)
```

Decrypted payload:

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

Figure 6: Example token request bound to an asymmetric key.

Figure 7 shows a request for a token where a previously communicated proof-of-possession key is only referenced. Note that the client performs a password based authentication in this example by submitting its client\_secret (see [Section 2.3.1 of \[RFC6749\]](#)). Note that this example uses the "req\_cnf" parameter from [\[I-D.ietf-ace-oauth-params\]](#).



```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "client_secret" : "mysecret234",
  "audience" : "valve424",
  "scope" : "read",
  "req_cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

Figure 7: Example request for an access token bound to a key reference.

Refresh tokens are typically not stored as securely as proof-of-possession keys in requesting clients. Proof-of-possession based refresh token requests MUST NOT request different proof-of-possession keys or different audiences in token requests. Refresh token requests can only use to request access tokens bound to the same proof-of-possession key and the same audience as access tokens issued in the initial token request.

#### **5.6.2. 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 response code equivalent to the CoAP response code 2.01 (Created). If client request was invalid, or not authorized, the AS returns an error response as described in [Section 5.6.3](#).

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 (see [Appendix D](#)). 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 is the Access Information. When using CBOR payloads, the content MUST be encoded as CBOR map, containing parameters as specified in [Section 5.1 of \[RFC6749\]](#), with the following additions and changes:

profile:



OPTIONAL. This indicates the profile that the client MUST use towards the RS. See [Section 5.6.4.3](#) for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

#### token\_type:

This parameter is OPTIONAL, as opposed to 'required' in [[RFC6749](#)]. By default implementations of this framework SHOULD assume that the token\_type is "pop". If a specific use case requires another token\_type (e.g., "Bearer") to be used then this parameter is REQUIRED.

Furthermore [[I-D.ietf-ace-oauth-params](#)] defines additional parameters that the AS MUST be able to use when responding to a request to the token endpoint.

Figure 8 summarizes the parameters that may be part of the Access Information. This does not include the additional parameters specified in [[I-D.ietf-ace-oauth-params](#)].

Parameter name	Specified in
access_token	<a href="#">RFC 6749</a>
token_type	<a href="#">RFC 6749</a>
expires_in	<a href="#">RFC 6749</a>
refresh_token	<a href="#">RFC 6749</a>
scope	<a href="#">RFC 6749</a>
state	<a href="#">RFC 6749</a>
error	<a href="#">RFC 6749</a>
error_description	<a href="#">RFC 6749</a>
error_uri	<a href="#">RFC 6749</a>
profile	[this document]

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key, which is defined in [[I-D.ietf-ace-oauth-params](#)].





```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
    (remainder of CWT omitted for brevity;
    CWT contains COSE_Key in the "cnf" claim)',
  "profile" : "coap_dtls",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkv8FJG+0nbc6mxCcQh'
    }
  }
}
```

Figure 9: Example AS response with an access token bound to a symmetric key.

### 5.6.3. 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:

- o When using CBOR the raw payload before being processed by the communication security protocol MUST be encoded as a CBOR map.
- o A response code equivalent to the CoAP code 4.00 (Bad Request) MUST be used for all error responses, except for `invalid_client` where a response code equivalent to the CoAP code 4.01 (Unauthorized) MAY be used under the same conditions as specified in [Section 5.2 of \[RFC6749\]](#).
- o The content type (for CoAP-based interactions) or media type (for HTTP-based interactions) "application/ace+cbor" MUST be used for the error response.
- o The parameters "error", "error\_description" and "error\_uri" MUST be abbreviated using the codes specified in Figure 12, when a CBOR encoding is used.
- o The error code (i.e., value of the "error" parameter) MUST be abbreviated as specified in Figure 10, when a CBOR encoding is used.



/-----+-----\   Name   CBOR Values    -----+-----	
invalid_request	1
invalid_client	2
invalid_grant	3
unauthorized_client	4
unsupported_grant_type	5
invalid_scope	6
unsupported_pop_key	7
incompatible_profiles	8
\-----+-----/	

Figure 10: CBOR abbreviations for common error codes

In addition to the error responses defined in OAuth 2.0, the following behavior MUST be implemented by the AS:

- o If the client submits an asymmetric key in the token request that the RS cannot process, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "unsupported\_pop\_key" defined in Figure 10.
- o If the client and the RS it has requested an access token for do not share a common profile, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "incompatible\_profiles" defined in Figure 10.

#### **5.6.4. Request and Response Parameters**

This section provides more detail about the new parameters that can be used in access token requests and responses, as well as abbreviations for more compact encoding of existing parameters and common parameter values.

##### **5.6.4.1. Grant Type**

The abbreviations in Figure 11 MUST be used in CBOR encodings instead of the string values defined in [[RFC6749](#)], if CBOR payloads are used.



/-----+-----+-----\		
Name	CBOR Value	Original Specification
-----+-----+-----		
password	0	<a href="#">RFC6749</a>
authorization_code	1	<a href="#">RFC6749</a>
client_credentials	2	<a href="#">RFC6749</a>
refresh_token	3	<a href="#">RFC6749</a>
\-----+-----+-----/		

Figure 11: CBOR abbreviations for common grant types

#### 5.6.4.2. Token Type

The "token\_type" parameter, defined in [[RFC6749](#)], allows the AS to indicate to the client which type of access token it is receiving (e.g., a bearer token).

This document registers the new value "pop" for the OAuth Access Token Types registry, specifying a proof-of-possession token. How the proof-of-possession by the client to the RS is performed MUST be specified by the profiles.

The values in the "token\_type" parameter MUST be CBOR text strings, if a CBOR encoding is used.

In this framework the "pop" value for the "token\_type" parameter is the default. The AS may, however, provide a different value.

#### 5.6.4.3. Profile

Profiles of this framework MUST define the communication protocol and the communication security protocol between the client and the RS. The security protocol MUST provide encryption, integrity and replay protection. It MUST also provide a binding between requests and responses. Furthermore profiles MUST define proof-of-possession methods, if they support proof-of-possession tokens.

A profile MUST specify an identifier that MUST be used to uniquely identify itself in the "profile" parameter. The textual representation of the profile identifier is just intended for human readability and MUST NOT be used in parameters and claims.

Profiles MAY define additional parameters for both the token request and the Access Information in the access token response in order to support negotiation or signaling of profile specific parameters.



#### 5.6.5. Mapping Parameters to CBOR

If CBOR encoding is used, all OAuth parameters in access token requests and responses MUST be mapped to CBOR types as specified in Figure 12, using the given integer abbreviation for the map keys.

Note that we have aligned the abbreviations corresponding to claims with the abbreviations defined in [RFC8392].

Note also that abbreviations from -24 to 23 have a 1 byte encoding size in CBOR. We have thus chosen to assign abbreviations in that range to parameters we expect to be used most frequently in constrained scenarios.

Name	CBOR Key	Value Type
access_token	1	byte string
scope	9	text or byte string
audience	18	text string
client_id	24	text string
client_secret	25	byte string
response_type	26	text string
redirect_uri	27	text string
state	28	text string
code	29	byte string
error	30	unsigned integer
error_description	31	text string
error_uri	32	text string
grant_type	33	unsigned integer
token_type	34	unsigned integer
expires_in	35	unsigned integer
username	36	text string
password	37	text string
refresh_token	38	byte string
profile	39	unsigned integer

Figure 12: CBOR mappings used in token requests

#### 5.7. The Introspection Endpoint

Token introspection [RFC7662] can be OPTIONALLY provided by the AS, and is then used by the RS and potentially the client to query the AS for metadata about a given token, e.g., validity or scope. Analogous to the protocol defined in RFC 7662 [RFC7662] for HTTP and JSON, this section defines adaptations to more constrained environments using





CBOR and leaving the choice of the application protocol to the profile.

Communication between the requesting entity and the introspection endpoint at the AS MUST be integrity protected and encrypted. The communication security protocol MUST also provide a binding between requests and responses. Furthermore the two interacting parties MUST perform mutual authentication. Finally the AS SHOULD verify that the requesting entity has the right to access introspection information about the provided token. Profiles of this framework that support introspection MUST specify how authentication and communication security between the requesting entity and the AS is implemented.

The default name of this endpoint in an url-path is '/introspect', however implementations are not required to use this name and can define their own instead.

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

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

#### **5.7.1. Introspection Request**

The requesting entity sends a POST request to the introspection endpoint at the AS, the profile MUST specify how the communication is protected. If CBOR is used, the payload MUST be encoded as a CBOR map with a "token" entry containing either the access token or a reference to the token (e.g., the cti). Further optional parameters representing additional context that is known by the requesting entity to aid the AS in its response MAY be included.

For CoAP-based interaction, all messages MUST use the content type "application/ace+cbor", while for HTTP-based interactions the equivalent media type "application/ace+cbor" MUST be used.

The same parameters are required and optional as in [Section 2.1 of RFC 7662](#) [RFC7662].

For example, Figure 13 shows a RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that object security based on OSCORE [[I-D.ietf-core-object-security](#)] is assumed in this example, therefore the Content-Format is "application/oscore". Figure 14 shows the decoded payload.



```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "introspect"
OSCORE: 0x09, 0x05, 0x25
Content-Format: "application/oscore"
Payload:
... COSE content ...
```

Figure 13: Example introspection request.

```
{
  "token" : b64'7gj0dXJQ43U',
  "token_type_hint" : "pop"
}
```

Figure 14: Decoded token.

### **5.7.2. Introspection Response**

If the introspection request is authorized and successfully processed, the AS sends a response with the response code equivalent to the CoAP code 2.01 (Created). If the introspection request was invalid, not authorized or couldn't be processed the AS returns an error response as described in [Section 5.7.3](#).

In a successful response, the AS encodes the response parameters in a map including with the same required and optional parameters as in [Section 2.2 of RFC 7662](#) [[RFC7662](#)] with the following addition:

profile OPTIONAL. This indicates the profile that the RS MUST use with the client. See [Section 5.6.4.3](#) for more details on the formatting of this parameter.

Furthermore [[I-D.ietf-ace-oauth-params](#)] defines more parameters that the AS MUST be able to use when responding to a request to the introspection endpoint.

For example, Figure 15 shows an AS response to the introspection request in Figure 13. Note that this example contains the "cnf" parameter defined in [[I-D.ietf-ace-oauth-params](#)].



```
Header: Created Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "active" : true,
  "scope" : "read",
  "profile" : "coap_dtls",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 15: Example introspection response.

### 5.7.3. 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 and CBOR is used the payload MUST be encoded as a CBOR map and the Content-Format "application/ace+cbor" MUST be used.
- o If the credentials used by the requesting entity (usually the RS) are invalid the AS MUST respond with the response code equivalent to the CoAP code 4.01 (Unauthorized) and use the required and optional parameters from [Section 5.2 in RFC 6749](#) [[RFC6749](#)].
- o If the requesting entity does not have the right to perform this introspection request, the AS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden). In this case no payload is returned.
- o The parameters "error", "error\_description" and "error\_uri" MUST be abbreviated using the codes specified in Figure 12.
- o The error codes MUST be abbreviated using the codes specified in Figure 10.

Note that a properly formed and authorized query for an inactive or otherwise invalid token does not warrant an error response by this specification. In these cases, the authorization server MUST instead



respond with an introspection response with the "active" field set to "false".

#### 5.7.4. Mapping Introspection parameters to CBOR

If CBOR is used, the introspection request and response parameters MUST be mapped to CBOR types as specified in Figure 16, using the given integer abbreviation for the map key.

Note that we have aligned abbreviations that correspond to a claim with the abbreviations defined in [RFC8392] and the abbreviations of parameters with the same name from [Section 5.6.5](#).

Parameter name	CBOR Key	Value Type
iss	1	text string
sub	2	text string
aud	3	text string
exp	4	integer or floating-point number
nbf	5	integer or floating-point number
iat	6	integer or floating-point number
cti	7	byte string
scope	9	text or byte string
active	10	True or False
token	12	byte string
client_id	24	text string
error	30	unsigned integer
error_description	31	text string
error_uri	32	text string
token_type_hint	33	text string
token_type	34	text string
username	36	text string
profile	39	unsigned integer

Figure 16: CBOR Mappings to Token Introspection Parameters.

#### 5.8. The Access Token

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

In order to facilitate offline processing of access tokens, this document uses the "cnf" claim from





[I-D.ietf-ace-cwt-proof-of-possession] and specifies the "scope" claim for JWT- and CWT-encoded 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 \[RFC6749\]](#), but in addition implementers MAY use byte strings as scope values, to achieve compact encoding of large scope elements. The meaning of a specific scope value is application specific and expected to be known to the RS running that application.

If the AS needs to convey a hint to the RS about which profile it should use to communicate with the client, the AS MAY include a "profile" claim in the access token, with the same syntax and semantics as defined in [Section 5.6.4.3](#).

#### **[5.8.1](#). The Authorization Information Endpoint**

The access token, containing authorization information and information about 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 a RESTful protocol such as CoAP. Profiles of this framework MAY define other methods for token transport.

The method consists of an authz-info endpoint, implemented by the RS. A client using this method MUST make a POST request to the authz-info endpoint at the RS with the access token in the payload. The RS receiving the token MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with 2.01 (Created). [Section 5.8.1.1](#) outlines how an RS MUST proceed to verify the validity of an access token.

The RS MUST be prepared to store at least one access token for future use. This is a difference to how access tokens are handled in OAuth 2.0, where the access token is typically sent along with each request, and therefore not stored at the RS.

This specification RECOMMENDS that an RS stores only one token per proof-of-possession key, meaning that an additional token linked to the same key will overwrite any existing token at the RS.

If the payload sent to the authz-info endpoint does not parse to a token, the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request).



The RS MAY make an introspection request to validate the token before responding to the POST request to the authz-info endpoint.

Profiles MUST specify whether the authz-info endpoint is protected, including whether error responses from this endpoint are protected. Note that since the token contains information that allow the client and the RS to establish a security context in the first place, mutual authentication may not be possible at this point.

The default name of this endpoint in an url-path is '/authz-info', however implementations are not required to use this name and can define their own instead.

A RS MAY use introspection on a token received through the authz-info endpoint, e.g. if the token is an opaque reference. Some transport protocols may provide a way to indicate that the RS is busy and the client should retry after an interval; this type of status update would be appropriate while the RS is waiting for an introspection response.

#### **5.8.1.1. Verifying an Access Token**

When an RS receives an access token, it MUST verify it before storing it. The details of token verification depends on various aspects, including the token encoding, the type of token, the security protection applied to the token, and the claims. The token encoding matters since the security wrapper differs between the token encodings. For example, a CWT token uses COSE while a JWT token uses JOSE. The type of token also has an influence on the verification procedure since tokens may be self-contained whereby token verification may happen locally at the RS while a token-by-reference requires further interaction with the authorization server, for example using token introspection, to obtain the claims associated with the token reference. Self-contained token MUST, at a minimum, be integrity protected but they MAY also be encrypted.

For self-contained tokens the RS MUST process the security protection of the token first, as specified by the respective token format. For CWT the description can be found in [RFC8392] and for JWT the relevant specification is [RFC7519]. This MUST include a verification that security protection (and thus the token) was generated by an AS that has the right to issue access tokens for this RS.

In case the token is communicated by reference the RS needs to obtain the claims first. When the RS uses token introspection the relevant specification is [RFC7662] with CoAP transport specified in [Section 5.7](#).



Errors may happen during this initial processing stage:

- o If token or claim verification fails, the RS MUST discard the token and, if this was an interaction with authz-info, return an error message with a response code equivalent to the CoAP code 4.01 (Unauthorized).
- o If the claims cannot be obtained the RS MUST discard the token and, in case of an interaction via the authz-info endpoint, return an error message with a response code equivalent to the CoAP code 4.00 (Bad Request).

Next, the RS MUST verify claims, if present, contained in the access token. Errors are returned when claim checks fail, in the order of priority of this list:

iss The issuer claim must identify an AS that has the authority to issue access tokens for the receiving RS. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.01 (Unauthorized).

exp The expiration date must be in the future. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.01 (Unauthorized). Note that the RS has to terminate access rights to the protected resources at the time when the tokens expire.

aud The audience claim must refer to an audience that the RS identifies with. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden).

scope The RS must recognize value of the scope claim. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request). The RS MAY provide additional information in the error response, to clarify what went wrong.

If the access token contains any other claims that the RS cannot process the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request). The RS MAY provide additional detail in the error response to clarify which claim couldn't be processed.

Note that the Subject (sub) claim cannot always be verified when the token is submitted to the RS since the client may not have authenticated yet. Also note that a counter for the expires\_in (exp) claim MUST be initialized when the RS first verifies this token.



When sending error responses, the RS MAY use the error codes from [Section 3.1 of \[RFC6750\]](#), to provide additional details to the client.

#### **5.8.1.2. Protecting the Authorization Information Endpoint**

As this framework can be used in RESTful environments, it is important to make sure that attackers cannot perform unauthorized requests on the auth-info endpoints, other than submitting access tokens.

Specifically it SHOULD NOT be possible to perform GET, DELETE or PUT on the authz-info endpoint and on it's children (if any).

The POST method SHOULD NOT be allowed on children of the authz-info endpoint.

The RS SHOULD implement rate limiting measures to mitigate attacks aiming to overload the processing capacity of the RS by repeatedly submitting tokens. For CoAP-based communication the RS could use the mechanisms from [\[RFC8516\]](#) to indicate that it is overloaded.

#### **5.8.2. Client Requests to the RS**

If an RS receives a request from a client, and the target resource requires authorization, the RS MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession for that token.

The response code MUST be 4.01 (Unauthorized) in case the client has not performed the proof-of-possession, or if RS has no valid access token for the client. If RS has an access token for the client but not for the resource that was requested, RS MUST reject the request with a 4.03 (Forbidden). If RS has an access token for the client but it does not cover the action that was requested on the resource, RS MUST reject the request with a 4.05 (Method Not Allowed).

Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a dumb Client optimistically tries to access a requested resource with any access token received from AS. As malicious clients could pretend to be C to determine C's privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the Client is authenticated.

Note: The RS MAY use introspection for timely validation of an access token, at the time when a request is presented.





Note: Matching the claims of the access token (e.g., scope) to a specific request is application specific.

If the request matches a valid token and the client has performed the proof-of-possession for that token, the RS continues to process the request as specified by the underlying application.

### **5.8.3. Token Expiration**

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

- o The token is a CWT and includes an "exp" claim and possibly the "nbf" claim. The RS verifies these by comparing them to values from its internal clock as defined in [\[RFC7519\]](#). In this case the RS's internal clock must reflect the current date and time, or at least be synchronized with the AS's clock. How this clock synchronization would be performed is out of scope for this specification.
- o The RS verifies the validity of the token by performing an introspection request as specified in [Section 5.7](#). 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 In order to support token expiration for devices that have no reliable way of synchronizing their internal clocks, this specification defines the following approach: The claim "exp\_i" ("expires in") can be used, to provide the RS with the lifetime of the token in seconds from the time the RS first receives the token. This approach is of course vulnerable to malicious clients holding back tokens they do not want to expire. Such an attack can only be prevented if the RS is able to communicate with the AS in some regular intervals, so that the AS can provide the RS with a list of expired tokens. The drawback of this mitigation is that the RS might as well use the communication with the AS to synchronize its internal clock.

If a token that authorizes a long running request such as a CoAP Observe [\[RFC7641\]](#) expires, the RS MUST send an error response with the response code equivalent to the CoAP code 4.01 (Unauthorized) to the client and then terminate processing the long running request.



#### 5.8.4. Key Expiration

The AS provides the client with key material that the RS uses. This can either be a common symmetric pop-key, or an asymmetric key used by the RS to authenticate towards the client. Since there is no metadata associated to those keys, the client has no way of knowing if these keys are still valid. This may lead to situations where the client sends requests containing sensitive information to the RS using a key that is expired and possibly in the hands of an attacker, or accepts responses from the RS that are not properly protected and could possibly have been forged by an attacker.

In order to prevent this, the client must assume that those keys are only valid as long as the related access token is. Since the access token is opaque to the client, one of the following methods **MUST** be used to inform the client about the validity of an access token:

- o The client knows a default validity period for all tokens it is using. This information could be provisioned to the client when it is registered at the AS, or published by the AS in a way that the client can query.
- o The AS informs the client about the token validity using the "expires\_in" parameter in the Access Information.
- o The client performs an introspection of the token. Although this is not explicitly forbidden, how exactly this is done is not currently specified for OAuth.

A client that is not able to obtain information about the expiration of a token **MUST NOT** use this token.

## 6. Security Considerations

Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [[RFC6749](#)] apply to this work. Furthermore [[RFC6819](#)] provides additional security considerations for OAuth which apply to IoT deployments as well. If the introspection endpoint is used, the security considerations from [[RFC7662](#)] also apply.

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest (MAC) or an Authenticated Encryption with Associated Data (AEAD) algorithm. Consequently, the token integrity protection **MUST** be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key, this symmetric



key MUST be encrypted by the authorization server so that only the resource server can decrypt it. Note that using an AEAD algorithm is preferable over using a MAC unless the message needs to be publicly readable.

If the token is intended for multiple recipients (i.e. an audience that is a group), integrity protection of the token with a symmetric key is not sufficient, since any of the recipients could modify the token undetected by the other recipients. Therefore a token with a multi-recipient audience MUST be protected with an asymmetric signature.

It is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. 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.

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client may obtain the proof-of-possession key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby completely negating proof-of-possession security. Profiles MUST specify how confidentiality protection is provided, and additional protection can be applied by encrypting the token, for example encryption of CWTs is specified in [Section 5.1 of \[RFC8392\]](#).

Developers MUST ensure that the ephemeral credentials (i.e., the private key or the session key) are not leaked to third parties. An adversary in possession of the ephemeral credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many constrained environments, since adversaries can often easily get physical access to the devices. This risk can also be mitigated to some extent by making sure that keys are refreshed more frequently.

If clients are capable of doing so, they should frequently request fresh access tokens, as this allows the AS to keep the lifetime of the tokens short. This allows the AS to use shorter proof-of-possession key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens, they SHOULD scope these access tokens to a specific permission.



### **6.1. Unprotected AS Request Creation Hints**

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the AS Request Creation Hints contained in an unprotected response from RS to an unauthorized request (see [Section 5.1.2](#)) are authentic. It is therefore advisable to provide C with a (possibly hard-coded) list of trustworthy authorization servers. AS Request Creation Hints referring to a URI not listed there would be ignored.

### **6.2. Minimal security requirements for communication**

This section summarizes the minimal requirements for the communication security of the different protocol interactions.

C-AS All communication between the client and the Authorization Server MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the client MUST be bound to the client's request to avoid attacks where the attacker swaps the intended response for an older one valid for a previous request. This requires that the client and the Authorization Server have previously exchanged either a shared secret, or their public keys in order to negotiate a secure communication. Furthermore the client MUST be able to determine whether an AS has the authority to issue access tokens for a certain RS. This can be done through pre-configured lists, or through an online lookup mechanism that in turn also must be secured.

RS-AS The communication between the Resource Server and the Authorization Server via the introspection endpoint MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the RS MUST be bound to the RS's request. This requires that the client and the Authorization Server have previously exchanged either a shared secret, or their public keys in order to negotiate a secure communication. Furthermore the RS MUST be able to determine whether an AS has the authority to issue access tokens itself. This is usually configured out of band, but could also be performed through an online lookup mechanism provided that it is also secured in the same way.

C-RS The initial communication between the client and the Resource Server can not be secured in general, since the RS is not in possession of an access token for that client, which would carry the necessary parameters. Certain security mechanisms (e.g. DTLS with server-side authentication via a certificate or a raw public key) can be possible and are RECOMMEND if supported by both parties. After the client has successfully transmitted the access token to the RS, a secure communication protocol MUST be





established between client and RS for the actual resource request. This protocol MUST provide encryption, integrity and replay protection as well as a binding between requests and responses. This requires that the client learned either the RS's public key or received a symmetric proof-of-possession key bound to the access token from the AS. The RS must have learned either the client's public key or a shared symmetric key from the claims in the token or an introspection request. Since ACE does not provide profile negotiation between C and RS, the client MUST have learned what profile the RS supports (e.g. from the AS or pre-configured) and initiate the communication accordingly.

### **6.3. Use of Nonces for Replay Protection**

The RS may add a nonce to the AS Request Creation Hints message sent as a response to an unauthorized request to ensure freshness of an Access Token subsequently presented to RS. While a time-stamp of some granularity would be sufficient to protect against replay attacks, using randomized nonce is preferred to prevent disclosure of information about RS's internal clock characteristics.

### **6.4. Combining profiles**

There may be use cases where different profiles of this framework are combined. For example, an MQTT-TLS profile is used between the client and the RS in combination with a CoAP-DTLS profile for interactions between the client and the AS. Ideally, profiles should be designed in a way that the security of system should not depend on the specific security mechanisms used in individual protocol interactions.

### **6.5. Unprotected Information**

Communication with the authz-info endpoint, as well as the various error responses defined in this framework all potentially include sending information over an unprotected channel. These messages may leak information to an adversary. For example errors responses for requests to the Authorization Information endpoint can reveal information about an otherwise opaque access token to an adversary who has intercepted this token.

As far as error messages are concerned, this framework is written under the assumption that, in general, the benefits of detailed error messages outweigh the risk due to information leakage. For particular use cases, where this assessment does not apply, detailed error messages can be replaced by more generic ones.



In some scenarios it may be possible to protect the communication with the authz-info endpoint (e.g. through DTLS with only server-side authentication). In cases where this is not possible this framework RECOMMENDS to use encrypted CWTs or opaque references and need to be subjected to introspection by the RS.

If the initial unauthorized resource request message (see [Section 5.1.1](#)) is used, the client MUST make sure that it is not sending sensitive content in this request. While GET and DELETE requests only reveal the target URI of the resource, while POST and PUT requests would reveal the whole payload of the intended operation.

### 6.6. Identifying audiences

The audience claim as defined in [[RFC7519](#)] and the equivalent "audience" parameter from [[I-D.ietf-oauth-token-exchange](#)] are intentionally vague on how to match the audience value to a specific RS. This is intended to allow application specific semantics to be used. This section attempts to give some general guidance for the use of audiences in constrained environments.

URLs are not a good way of identifying mobile devices that can switch networks and thus be associated with new URLs. If the audience represents a single RS, and asymmetric keys are used, the RS can be uniquely identified by a hash of its public key. If this approach is used this framework RECOMMENDS to apply the procedure from [section 3 of \[RFC6920\]](#).

If the audience addresses a group of resource servers, the mapping of group identifier to individual RS has to be provisioned to each RS before the group-audience is usable. Managing dynamic groups could be an issue, if the RS is not always reachable when the group memberships change. Furthermore issuing access tokens bound to symmetric proof-of-possession keys that apply to a group-audience is problematic, as an RS that is in possession of the access token can impersonate the client towards the other RSs that are part of the group. It is therefore NOT RECOMMENDED to issue access tokens bound to a group audience and symmetric proof-of possession keys.

Even the client must be able to determine the correct values to put into the "audience" parameter, in order to obtain a token for the intended RS. Errors in this process can lead to the client inadvertently communicating with the wrong RS. The correct values for "audience" can either be provisioned to the client as part of its configuration, or provided by the RS as part of the "AS Request Creation Hints" [Section 5.1.2](#) or dynamically looked up by the client



in some directory. In the latter case the integrity and correctness of the directory data must be assured.

#### **6.7. Denial of service against or with Introspection**

The optional introspection mechanism provided by OAuth and supported in the ACE framework allows for two types of attacks that need to be considered by implementers.

First an attacker could perform a denial of service attack against the introspection endpoint at the AS in order to prevent validation of access tokens. To mitigate this attack, an RS that is configured to use introspection **MUST NOT** allow access based on a token for which it couldn't reach the introspection endpoint.

Second an attacker could use the fact that an RS performs introspection to perform a denial of service attack against that RS by repeatedly sending tokens to its authz-info endpoint that require an introspection call. RS can mitigate such attacks by implementing a rate limit on how many introspection requests they perform in a given time interval and rejecting incoming requests to authz-info for a certain amount of time, when that rate limit has been reached.

### **7. Privacy Considerations**

Implementers and users should be aware of the privacy implications of the different possible deployments of this framework.

The AS is in a very central position and can potentially learn sensitive information about the clients requesting access tokens. If the client credentials grant is used, the AS can track what kind of access the client intends to perform. With other grants this can be prevented by the Resource Owner. To do so, the resource owner needs to bind the grants it issues to anonymous, ephemeral credentials that do not allow the AS to link different grants and thus different access token requests by the same client.

If access tokens are only integrity protected and not encrypted, they may reveal information to attackers listening on the wire, or able to acquire the access tokens in some other way. In the case of CWTs the token may, e.g., reveal the audience, the scope and the confirmation method used by the client. The latter may reveal the identity of the device or application running the client. This may be linkable to the identity of the person using the client (if there is a person and not a machine-to-machine interaction).

Clients using asymmetric keys for proof-of-possession should be aware of the consequences of using the same key pair for proof-of-



possession towards different RSs. A set of colluding RSs or an attacker able to obtain the access tokens will be able to link the requests, or even to determine the client's identity.

An unprotected response to an unauthorized request (see [Section 5.1.2](#)) may disclose information about RS and/or its existing relationship with C. It is advisable to include as little information as possible in an unencrypted response. Means of encrypting communication between C and RS already exist, more detailed information may be included with an error response to provide C with sufficient information to react on that particular error.

## **8. IANA Considerations**

### **[8.1.](#) ACE Authorization Server Request Creation Hints**

This specification establishes the IANA "ACE Authorization Server Request Creation Hints" registry. The registry has been created to use the "Expert Review Required" registration procedure [[RFC8126](#)]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of the registry are:

Name The name of the parameter

CBOR Key CBOR map key for the parameter. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Value Type The CBOR data types allowable for the values of this parameter.

Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.

This registry will be initially populated by the values in Figure 2. The Reference column for all of these entries will be this document.





## 8.2. OAuth Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry defined in [\[RFC6749\]](#).

- o Error name: "unsupported\_pop\_key"
- o Error usage location: AS token endpoint error response
- o Related protocol extension: The ACE framework [this document]
- o Change Controller: IESG
- o Specification document(s): [Section 5.6.3](#) of [this document]
  
- o Error name: "incompatible\_profiles"
- o Error usage location: AS token endpoint error response
- o Related protocol extension: The ACE framework [this document]
- o Change Controller: IESG
- o Specification document(s): [Section 5.6.3](#) of [this document]

## 8.3. OAuth Error Code CBOR Mappings Registry

This specification establishes the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [\[RFC8126\]](#). It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of the registry are:

Name The OAuth Error Code name, refers to the name in [Section 5.2. of \[RFC6749\]](#), e.g., "invalid\_request".

CBOR Value CBOR abbreviation for this error code. Different ranges of values use different registration policies [\[RFC8126\]](#). Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.

This registry will be initially populated by the values in Figure 10. The Reference column for all of these entries will be this document.

## 8.4. OAuth Grant Type CBOR Mappings

This specification establishes the IANA "OAuth Grant Type CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [\[RFC8126\]](#). It should be



noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

Name The name of the grant type as specified in [Section 1.3 of \[RFC6749\]](#).

CBOR Value CBOR abbreviation for this grant type. Different ranges of values use different registration policies [\[RFC8126\]](#). Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.

Original Specification This contains a pointer to the public specification of the grant type, if one exists.

This registry will be initially populated by the values in Figure 11. The Reference column for all of these entries will be this document.

### **8.5. OAuth Access Token Types**

This section registers the following new token type in the "OAuth Access Token Types" registry [\[IANA.OAuthAccessTokenTypes\]](#).

- o Name: "PoP"
- o Change Controller: IETF
- o Reference: [this document]

### **8.6. OAuth Access Token Type CBOR Mappings**

This specification established the IANA "OAuth Access Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [\[RFC8126\]](#). It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

Name The name of token type as registered in the OAuth Access Token Types registry, e.g., "Bearer".

CBOR Value CBOR abbreviation for this token type. Different ranges of values use different registration policies [\[RFC8126\]](#). Integer values from -256 to 255 are designated as Standards Action.



Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

**Reference** This contains a pointer to the public specification of the OAuth token type abbreviation, if one exists.

**Original Specification** This contains a pointer to the public specification of the grant type, if one exists.

#### **8.6.1. Initial Registry Contents**

- o Name: "Bearer"
- o Value: 1
- o Reference: [this document]
- o Original Specification: [[RFC6749](#)]
  
- o Name: "pop"
- o Value: 2
- o Reference: [this document]
- o Original Specification: [this document]

#### **8.7. ACE Profile Registry**

This specification establishes the IANA "ACE Profile" registry. The registry has been created to use the "Expert Review Required" registration procedure [[RFC8126](#)]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

**Name** The name of the profile, to be used as value of the profile attribute.

**Description** Text giving an overview of the profile and the context it is developed for.

**CBOR Value** CBOR abbreviation for this profile name. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

**Reference** This contains a pointer to the public specification of the profile abbreviation, if one exists.

This registry will be initially empty and will be populated by the registrations from the ACE framework profiles.



### **8.8. OAuth Parameter Registration**

This specification registers the following parameter in the "OAuth Parameters" registry [[IANA.OAuthParameters](#)]:

- o Name: "profile"
- o Parameter Usage Location: token response
- o Change Controller: IESG
- o Reference: [Section 5.6.4.3](#) of [this document]

### **8.9. OAuth Parameters CBOR Mappings Registry**

This specification establishes the IANA "OAuth Parameters CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [[RFC8126](#)]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

Name	The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".
CBOR Key	CBOR map key for this parameter. Different ranges of values use different registration policies [ <a href="#">RFC8126</a> ]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
Value Type	The allowable CBOR data types for values of this parameter.
Reference	This contains a pointer to the public specification of the parameter abbreviation, if one exists.

This registry will be initially populated by the values in Figure 12. The Reference column for all of these entries will be this document.

Note that the mappings of parameters corresponding to claim names intentionally coincide with the CWT claim name mappings from [[RFC8392](#)].

### **8.10. OAuth Introspection Response Parameter Registration**

This specification registers the following parameter in the OAuth Token Introspection Response registry [[IANA.TokenIntrospectionResponse](#)].





- 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 Reference: [Section 5.7.2](#) of [this document]

### **8.11. OAuth Token Introspection Response CBOR Mappings Registry**

This specification establishes the IANA "OAuth Token Introspection Response CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [[RFC8126](#)]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

Name The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client\_id".

CBOR Key CBOR map key for this parameter. Different ranges of values use different registration policies [[RFC8126](#)]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Value Type The allowable CBOR data types for values of this parameter.

Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.

This registry will be initially populated by the values in Figure 16. The Reference column for all of these entries will be this document.

Note that the mappings of parameters corresponding to claim names intentionally coincide with the CWT claim name mappings from [[RFC8392](#)].

### **8.12. JSON Web Token Claims**

This specification registers the following new claims in the JSON Web Token (JWT) registry of JSON Web Token Claims [[IANA.JsonWebTokenClaims](#)]:

- o Claim Name: "scope"
- o Claim Description: The scope of an access token as defined in [[RFC6749](#)].
- o Change Controller: IESG



- o Reference: [Section 5.8](#) of [this document]
- o Claim Name: "profile"
- o Claim Description: The profile a token is supposed to be used with.
- o Change Controller: IESG
- o Reference: [Section 5.8](#) of [this document]
- o Claim Name: "exp"
- o Claim Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker form of token expiration for devices that cannot synchronize their internal clocks.
- o Change Controller: IESG
- o Reference: [Section 5.8.3](#) of [this document]

### **[8.13](#). CBOR Web Token Claims**

This specification registers the following new claims in the "CBOR Web Token (CWT) Claims" registry [[IANA.CborWebTokenClaims](#)].

- o Claim Name: "scope"
- o Claim Description: The scope of an access token as defined in [[RFC6749](#)].
- o JWT Claim Name: scope
- o Claim Key: TBD (suggested: 9)
- o Claim Value Type(s): byte string or text string
- o Change Controller: IESG
- o Specification Document(s): [Section 5.8](#) of [this document]
- o Claim Name: "profile"
- o Claim Description: The profile a token is supposed to be used with.
- o JWT Claim Name: profile
- o Claim Key: TBD (suggested: 39)
- o Claim Value Type(s): integer
- o Change Controller: IESG
- o Specification Document(s): [Section 5.8](#) of [this document]
- o Claim Name: "exp"
- o Claim Description: The expiration time of a token measured from when it was received at the RS in seconds.
- o JWT Claim Name: exp
- o Claim Key: TBD (suggested: 41)
- o Claim Value Type(s): integer
- o Change Controller: IESG
- o Specification Document(s): [Section 5.8](#) of [this document]



#### **8.14. Media Type Registrations**

This specification registers the 'application/ace+cbor' media type for messages of the protocols defined in this document carrying parameters encoded in CBOR. This registration follows the procedures specified in [[RFC6838](#)].

Type name: application

Subtype name: ace+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as CBOR map containing the protocol parameters defined in [this document].

Security considerations: See [Section 6](#) of this document.

Interoperability considerations: n/a

Published specification: [this document]

Applications that use this media type: The type is used by authorization servers, clients and resource servers that support the ACE framework as specified in [this document].

Additional information:

Magic number(s): n/a

File extension(s): .ace

Macintosh file type code(s): n/a

Person & email address to contact for further information: Ludwig Seitz <ludwig.seitz@ri.se>

Intended usage: COMMON

Restrictions on usage: None

Author: Ludwig Seitz <ludwig.setiz@ri.se>

Change controller: IESG



### **8.15. CoAP Content-Format Registry**

This specification registers the following entry to the "CoAP Content-Formats" registry:

Media Type: application/ace+cbor

Encoding

ID: 19

Reference: [this document]

### **8.16. Expert Review Instructions**

All of the IANA registries established in this document are defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered, and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.
- o Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- o Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.





- o Since a high degree of overlap is expected between these registries and the contents of the OAuth parameters [[IANA.OAuthParameters](#)] registries, experts should require new registrations to maintain a reasonable level of alignment with parameters from OAuth that have comparable functionality.

## **9. Acknowledgments**

This document is a product of the ACE working group of the IETF.

Thanks to Eve Maler for her contributions to the use of OAuth 2.0 and UMA in IoT scenarios, Robert Taylor for his discussion input, and Malisa Vucinic for his input on the predecessors of this proposal.

Thanks to the authors of [draft-ietf-oauth-pop-key-distribution](#), from where large parts of the security considerations were copied.

Thanks to Stefanie Gerdes, Olaf Bergmann, and Carsten Bormann for contributing their work on AS discovery from [draft-gerdes-ace-dcaf-authorize](#) (see [Section 5.1](#)).

Thanks to Jim Schaad and Mike Jones for their comprehensive reviews.

Thanks to Benjamin Kaduk for his input on various questions related to this work.

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

## **10. References**

### **10.1. Normative References**

[I-D.ietf-ace-cwt-proof-of-possession]

Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", [draft-ietf-ace-cwt-proof-of-possession-05](#) (work in progress), November 2018.

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

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", [draft-ietf-ace-oauth-params-04](#) (work in progress), February 2019.

[I-D.ietf-oauth-token-exchange]

Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", [draft-ietf-oauth-token-exchange-16](#) (work in progress), October 2018.



[IANA.CborWebTokenClaims]

IANA, "CBOR Web Token (CWT) Claims",  
<<https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>>.

[IANA.JsonWebTokenClaims]

IANA, "JSON Web Token Claims",  
<<https://www.iana.org/assignments/jwt/jwt.xhtml#claims>>.

[IANA.OAuthAccessTokenTypes]

IANA, "OAuth Access Token Types",  
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-types>>.

[IANA.OAuthParameters]

IANA, "OAuth Parameters",  
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#parameters>>.

[IANA.TokenIntrospectionResponse]

IANA, "OAuth Token Introspection Response",  
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-introspection-response>>.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, [RFC 3986](#), DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.

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

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", [RFC 6750](#), DOI 10.17487/RFC6750, October 2012, <<https://www.rfc-editor.org/info/rfc6750>>.



- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", [BCP 13](#), [RFC 6838](#), DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", [RFC 6920](#), DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", [RFC 7252](#), DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", [RFC 7662](#), DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 8126](#), DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", [RFC 8152](#), DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", [RFC 8392](#), DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

## **10.2. Informative References**

- [I-D.erdman-ace-rpcc]  
Seitz, L. and S. Erdtman, "Raw-Public-Key and Pre-Shared-Key as OAuth client credentials", [draft-erdman-ace-rpcc-02](#) (work in progress), October 2017.
- [I-D.ietf-core-object-security]  
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", [draft-ietf-core-object-security-15](#) (work in progress), August 2018.



[I-D.ietf-oauth-device-flow]

Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Flow for Browserless and Input Constrained Devices", [draft-ietf-oauth-device-flow-14](#) (work in progress), January 2019.

[I-D.ietf-tls-dtls13]

Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", [draft-ietf-tls-dtls13-30](#) (work in progress), November 2018.

[Margi10impact]

Margi, C., de Oliveira, B., de Sousa, G., Simplicio Jr, M., Barreto, P., Carvalho, T., Naeslund, M., and R. Gold, "Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds", Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), August 2010.

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

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

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", [RFC 6819](#), DOI 10.17487/RFC6819, January 2013, <<https://www.rfc-editor.org/info/rfc6819>>.

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", [RFC 7228](#), DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

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





- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", [RFC 7519](#), DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", [RFC 7521](#), DOI 10.17487/RFC7521, May 2015, <<https://www.rfc-editor.org/info/rfc7521>>.
- [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", [RFC 7591](#), DOI 10.17487/RFC7591, July 2015, <<https://www.rfc-editor.org/info/rfc7591>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", [RFC 7641](#), DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", [RFC 7744](#), DOI 10.17487/RFC7744, January 2016, <<https://www.rfc-editor.org/info/rfc7744>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", [RFC 7959](#), DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", [BCP 212](#), [RFC 8252](#), DOI 10.17487/RFC8252, October 2017, <<https://www.rfc-editor.org/info/rfc8252>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, [RFC 8259](#), DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", [RFC 8414](#), DOI 10.17487/RFC8414, June 2018, <<https://www.rfc-editor.org/info/rfc8414>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.



[RFC8516] Keraenen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", [RFC 8516](#), DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.

## **Appendix A. Design Justification**

This section provides further insight into the design decisions of the solution documented in this document. [Section 3](#) lists several building blocks and briefly summarizes their importance. The justification for offering some of those building blocks, as opposed to using OAuth 2.0 as is, is given below.

Common IoT constraints are:

Low Power Radio:

Many IoT devices are equipped with a small battery which needs to last for a long time. For many constrained wireless devices, the highest energy cost is associated to transmitting or receiving messages (roughly by a factor of 10 compared to AES) [[Margi10impact](#)]. It is therefore important to keep the total communication overhead low, including minimizing the number and size of messages sent and received, which has an impact of choice on the message format and protocol. By using CoAP over UDP and CBOR encoded messages, some of these aspects are addressed. Security protocols contribute to the communication overhead and can, in some cases, be optimized. For example, authentication and key establishment may, in certain cases where security requirements allow, be replaced by provisioning of security context by a trusted third party, using transport or application layer security.

Low CPU Speed:

Some IoT devices are equipped with processors that are significantly slower than those found in most current devices on the Internet. This typically has implications on what timely cryptographic operations a device is capable of performing, which in turn impacts, e.g., protocol latency. Symmetric key cryptography may be used instead of the computationally more expensive public key cryptography where the security requirements so 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 may not want to give access to the data collected by their IoT device or to functions it may offer to third parties. Since the classical approach of requesting permissions from end users via a rich user interface does not work in many IoT deployment scenarios, these functions need to be delegated to user-controlled devices that are better suitable for such tasks, such as smart phones and tablets.

#### Communication Constraints:

In certain constrained settings an IoT device may not be able to communicate with a given device at all times. Devices may be sleeping, or just disconnected from the Internet because of general lack of connectivity in the area, for cost reasons, or for security reasons, e.g., to avoid an entry point for Denial-of-Service attacks.

The communication interactions this framework builds upon (as shown graphically in Figure 1) may be accomplished using a variety of different protocols, and not all parts of the message flow are used in all applications due to the communication constraints. Deployments making use of CoAP are expected, but not limited to, other protocols such as HTTP, HTTP/2 or other specific protocols, such as Bluetooth Smart communication, that do not necessarily use IP could also be used. The latter raises the need for application layer security over the various interfaces.

In the light of these constraints we have made the following design decisions:

CBOR, COSE, CWT:



This framework RECOMMENDS the use of CBOR [[RFC7049](#)] as data format. Where CBOR data needs to be protected, the use of COSE [[RFC8152](#)] is RECOMMENDED. Furthermore where self-contained tokens are needed, this framework RECOMMENDS the use of CWT [[RFC8392](#)]. These measures aim at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

#### CoAP:

This framework RECOMMENDS the use of CoAP [[RFC7252](#)] instead of HTTP. This does not preclude the use of other protocols specifically aimed at constrained devices, like, e.g., Bluetooth Low Energy (see [Section 3.2](#)). This aims again at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

#### Access Information:

This framework defines the name "Access Information" for data concerning the RS that the AS returns to the client in an access token response (see [Section 5.6.2](#)). This aims at enabling scenarios, where a powerful client, supporting multiple profiles, needs to interact with a RS for which it does not know the supported profiles and the raw public key.

#### Proof-of-Possession:

This framework makes use of proof-of-possession tokens, using the "cnf" claim [[I-D.ietf-ace-cwt-proof-of-possession](#)]. A semantically and syntactically identical request and response parameter is defined for the token endpoint, to allow requesting and stating confirmation keys. This aims at making token theft harder. Token theft is specifically relevant in constrained use cases, as communication often passes through middle-boxes, which could be able to steal bearer tokens and use them to gain unauthorized access.

#### Auth-Info endpoint:

This framework introduces a new way of providing access tokens to a RS by exposing a authz-info endpoint, to which access tokens can be POSTed. This aims at reducing the size of the request message and the code complexity at the RS. The size of the request message is problematic, since many constrained protocols have severe message size limitations at the physical layer (e.g., in





the order of 100 bytes). This means that larger packets get fragmented, which in turn combines badly with the high rate of packet loss, and the need to retransmit the whole message if one packet gets lost. Thus separating sending of the request and sending of the access tokens helps to reduce fragmentation.

#### Client Credentials Grant:

This framework RECOMMENDS the use of the client credentials grant for machine-to-machine communication use cases, where manual intervention of the resource owner to produce a grant token is not feasible. The intention is that the resource owner would instead pre-arrange authorization with the AS, based on the client's own credentials. The client can then (without manual intervention) obtain access tokens from the AS.

#### Introspection:

This framework RECOMMENDS the use of access token introspection in cases where the client is constrained in a way that it can not easily obtain new access tokens (i.e. it has connectivity issues that prevent it from communicating with the AS). In that case this framework RECOMMENDS the use of a long-term token, that could be a simple reference. The RS is assumed to be able to communicate with the AS, and can therefore perform introspection, in order to learn the claims associated with the token reference. The advantage of such an approach is that the resource owner can change the claims associated to the token reference without having to be in contact with the client, thus granting or revoking access rights.

## [Appendix B](#). Roles and Responsibilities

### Resource Owner

- \* Make sure that the RS is registered at the AS. This includes making known to the AS which profiles, token\_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 that is in charge of the RS.
- \* If the client-credentials grant is used, make sure that the AS has the necessary, up-to-date, access control policies for the RS.

### Requesting Party



- \* Make sure that the client is provisioned the necessary credentials to authenticate to the AS.
- \* Make sure that the client is configured to follow the security requirements of the Requesting Party when issuing requests (e.g., minimum communication security requirements, trust anchors).
- \* Register the client at the AS. This includes making known to the AS which profiles, token\_types, and key types (symmetric/asymmetric) the client.

#### Authorization Server

- \* Register the RS and manage corresponding security contexts.
- \* Register clients and authentication credentials.
- \* Allow Resource Owners to configure and update access control policies related to their registered RSs.
- \* Expose the token endpoint to allow clients to request tokens.
- \* Authenticate clients that wish to request a token.
- \* Process a token request using the authorization policies configured for the RS.
- \* Optionally: Expose the introspection endpoint that allows RS's to submit token introspection requests.
- \* If providing an introspection endpoint: Authenticate RSs that wish to get an introspection response.
- \* If providing an introspection endpoint: Process token introspection requests.
- \* Optionally: Handle token revocation.
- \* Optionally: Provide discovery metadata. See [[RFC8414](#)]
- \* Optionally: Handle refresh tokens.

#### Client

- \* Discover the AS in charge of the RS that is to be targeted with a request.
- \* Submit the token request (see step (A) of Figure 1).
  - + Authenticate to the AS.
  - + Optionally (if not pre-configured): Specify which RS, which resource(s), and which action(s) the request(s) will target.
  - + If raw public keys (rpk) or certificates are used, make sure the AS has the right rpk or certificate for this client.
- \* Process the access token and Access Information (see step (B) of Figure 1).
  - + Check that the Access Information provides the necessary security parameters (e.g., PoP key, information on communication security protocols supported by the RS).
  - + Safely store the proof-of-possession key.



- + If provided by the AS: Safely store the refresh token.
- \* Send the token and request to the RS (see step (C) of Figure 1).
  - + Authenticate towards the RS (this could coincide with the proof of possession process).
  - + Transmit the token as specified by the AS (default is to the authz-info endpoint, alternative options are specified by profiles).
  - + Perform the proof-of-possession procedure as specified by the profile in use (this may already have been taken care of through the authentication procedure).
- \* Process the RS response (see step (F) of Figure 1) of the RS.

#### Resource Server

- \* Expose a way to submit access tokens. By default this is the authz-info endpoint.
- \* Process an access token.
  - + Verify the token is from a recognized AS.
  - + 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.
- \* Safely store credentials such as raw public keys for authentication or proof-of-possession keys linked to access tokens.

#### [Appendix C](#). Requirements on Profiles

This section lists the requirements on profiles of this framework, for the convenience of profile designers.



- o Specify the communication protocol the client and RS must use (e.g., CoAP). [Section 5](#) and [Section 5.6.4.3](#)
- o Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS over CoAP). This must provide encryption, integrity and replay protection. [Section 5.6.4.3](#)
- o Specify how the client and the RS mutually authenticate. [Section 4](#)
- o Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol. [Section 5.6.4.2](#)
- o Specify a unique profile identifier. [Section 5.6.4.3](#)
- o If introspection is supported: Specify the communication and security protocol for introspection. [Section 5.7](#)
- o Specify the communication and security protocol for interactions between client and AS. This must provide encryption, integrity protection, replay protection and a binding between requests and responses. [Section 5](#) and [Section 5.6](#)
- o Specify how/if the authz-info endpoint is protected, including how error responses are protected. [Section 5.8.1](#)
- o Optionally define other methods of token transport than the authz-info endpoint. [Section 5.8.1](#)

#### [Appendix D](#). Assumptions on AS knowledge about C and RS

This section lists the assumptions on what an AS should know about a client and a RS in order to be able to respond to requests to the token and introspection endpoints. How this information is established is out of scope for this document.

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





## **Appendix E. Deployment Examples**

There is a large variety of IoT deployments, as is indicated in [Appendix A](#), and this section highlights a few common variants. This section is not normative but illustrates how the framework can be applied.

For each of the deployment variants, there are a number of possible security setups between clients, resource servers and authorization servers. The main focus in the following subsections is on how authorization of a client request for a resource hosted by a RS is performed. This requires 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.

### **E.1. Local Token Validation**

In this scenario, the case where the resource server is offline is considered, i.e., it is not connected to the AS at the time of the access request. This access procedure involves steps A, B, C, and F of Figure 1.

Since the resource server must be able to verify the access token locally, self-contained access tokens must be used.

This example shows the interactions between a client, the authorization server and a temperature sensor acting as a resource server. Message exchanges A and B are shown in Figure 17.

A: The client first generates a public-private key pair used for communication security with the RS. The client sends the POST request to the token endpoint at the AS. The security of this request can be transport or application layer. It is up to the communication security profile to define. In the example transport 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 access token and Access Information. The PoP access token contains the public key of the client, and the Access Information contains the public key of the RS. For communication security this example uses DTLS RawPublicKey between the client and the RS. The issued token will have a short



validity time, i.e., "exp" close to "iat", 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 it is assumed that the client knows what resource it wants to access, and is therefore able to request specific audience and scope claims for the access token.

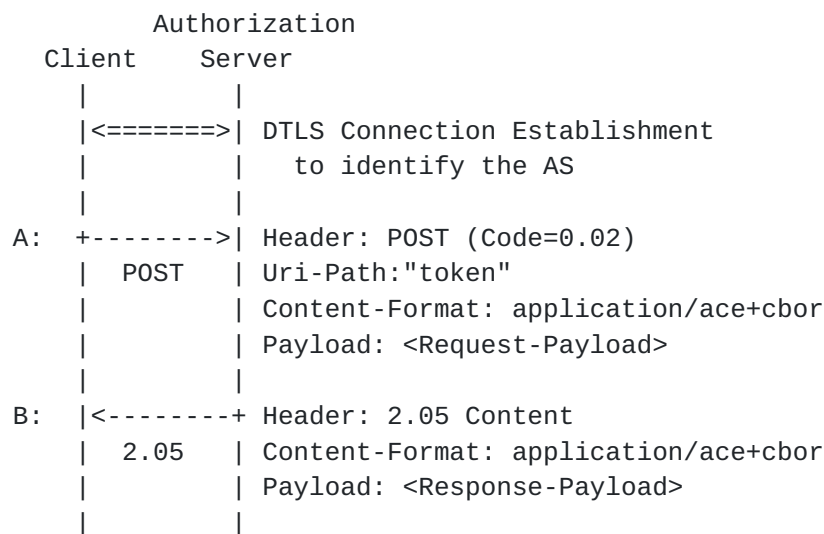


Figure 17: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 18 Note that the parameter "rs\_cnf" from [\[I-D.ietf-ace-oauth-params\]](#) is used to inform the client about the resource server's public key.



Request-Payload :

```
{
  "audience" : "tempSensorInLivingRoom",
  "client_id" : "myclient",
  "client_secret" : "qwerty"
  "req_cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}
```

Response-Payload :

```
{
  "access_token" : b64'SlAV32hkKG ...',
  "rs_cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'MKBCTNIckUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y" : b64'4Et16SRW2YiLUrN5vfvVHuHp7x8PxltmWWlbbM4IFyM'
    }
  }
}
```

Figure 18: Request and Response Payload Details.

The content of the access token is shown in Figure 19.



```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

```

Figure 19: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 20 - Figure 21.

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

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

Figure 20: Access Token provisioning to RS

The client and the RS runs the DTLS handshake using the raw public keys established in step B and C.

The client sends 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.





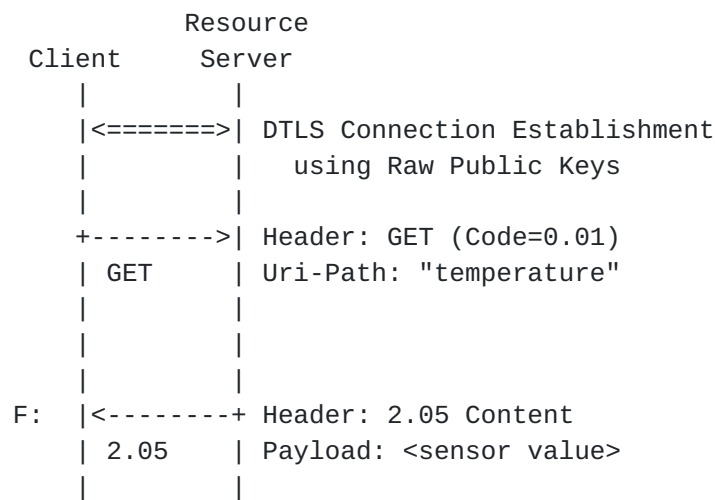


Figure 21: Resource Request and Response protected by DTLS.

## E.2. Introspection Aided Token Validation

In this deployment scenario it is assumed that a client is not able to access the AS at the time of the access request, whereas the RS is assumed to be connected to the back-end infrastructure. Thus the RS can make use of token introspection. This access procedure involves steps A-F of Figure 1, but assumes steps A and B have been carried out during a phase when the client had connectivity to AS.

Since the client is assumed to be offline, at least for a certain period of time, a pre-provisioned access token has to be long-lived. Since the client is constrained, the token will not be self contained (i.e. not a CWT) but instead just a reference. The resource server uses its connectivity to learn about the claims associated to the access token by using introspection, which is shown in the example below.

In the example interactions between an offline client (key fob), a RS (online lock), and an AS is shown. It is assumed that there is a provisioning step where the client has access to the AS. This corresponds to message exchanges A and B which are shown in Figure 22.

Authorization consent from the resource owner can be pre-configured, but it can also be provided via an interactive flow with the resource owner. An example of this for the key fob case could be that the resource owner has a connected car, he buys a generic key that he wants to use with the car. To authorize the key fob he connects it to his computer that then provides the UI for the device. After that OAuth 2.0 implicit flow can be used to authorize the key for his car at the the car manufacturers AS.



Note: In this example the client does not know the exact door it will be used to access since the token request is not send at the time of access. So the scope and audience parameters are set quite wide to start with and new values different form the original once can be returned from introspection later on.

A: The client sends the request using POST to the token endpoint 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 an 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 Access Information, the latter containing a symmetric key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key is used as the PreSharedKey.

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

Figure 22: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 23.



Request-Payload:

```
{
  "client_id" : "keyfob",
  "client_secret" : "qwerty"
}
```

Response-Payload:

```
{
  "access_token" : b64'VGZzdCB0b2t1bg==',
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsawMga2V5IGlk',
      "kty" : "oct",
      "alg" : "HS256",
      "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzzff5oRJxl-IXRtztJ6uE'
    }
  }
}
```

Figure 23: Request and Response Payload for C offline

The access token in this case is just an opaque byte string referencing the authorization information at the AS.

C: Next, the client POSTs the access token to the authz-info endpoint in the RS. This is a plain CoAP request, i.e., no DTLS between client and RS. Since the token is an opaque string, the RS cannot verify it on its own, and thus defers to respond the client with a status code until after step E.

D: The RS forwards the token to the introspection 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).



Client	Resource Server
C: +----->	Header: POST (T=CON, Code=0.02)
POST	Uri-Path: "authz-info"
	Payload: b64'VGZzdCB0b2t1bg=='
	Authorization Server
D: +----->	Header: POST (Code=0.02)
POST	Uri-Path: "introspect"
	Content-Format: "application/ace+cbor"
	Payload: <Request-Payload>
E: <-----+	Header: 2.05 Content
2.05	Content-Format: "application/ace+cbor"
	Payload: <Response-Payload>
<-----+	Header: 2.01 Created
2.01	

Figure 24: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 25.

Request-Payload:

```
{
  "token" : b64'VGZzdCB0b2t1bg==',
  "client_id" : "FrontDoor",
  "client_secret" : "ytrewq"
}
```

Response-Payload:

```
{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1311280970,
  "cnf" : {
    "kid" : b64'c29tZSBwdWJsawMga2V5IGlk'
  }
}
```

Figure 25: Request and Response Payload for Introspection





The client uses the symmetric PoP key to establish a DTLS PreSharedKey secure connection to the RS. The CoAP request PUT is sent to the uri-path /state on the RS, changing the state of the door to locked.

F: The RS responds with a appropriate over the secure DTLS channel.

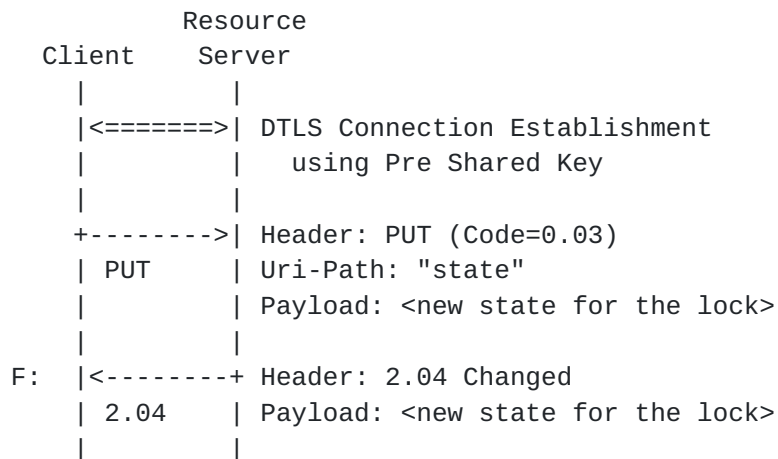


Figure 26: Resource request and response protected by OSCORE

## Appendix F. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

### F.1. Verion -20 to 21

- o Added text about expiration of RS keys.

### F.2. Verion -19 to 20

- o Replaced "req\_aud" with "audience" from the OAuth token exchange draft.
- o Updated examples to remove unnecessary elements.

### F.3. Version -18 to -19

- o Added definition of "Authorization Information".
- o Explicitly state that ACE allows encoding refresh tokens in binary format in addition to strings.
- o Renamed "AS Information" to "AS Request Creation Hints" and added the possibility to specify req\_aud and scope as hints.
- o Added the "kid" parameter to AS Request Creation Hints.
- o Added security considerations about the integrity protection of tokens with multi-RS audiences.



- o Renamed IANA registries mapping OAuth parameters to reflect the mapped registry.
- o Added JWT claim names to CWT claim registrations.
- o Added expert review instructions.
- o Updated references to TLS from 1.2 to 1.3.

#### **F.4. Version -17 to -18**

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

#### **F.5. Version -16 to -17**

- o Added references to (D)TLS 1.3.
- o Added requirement that responses are bound to requests.
- o Specify that grant\_type is OPTIONAL in C2AS requests (as opposed to REQUIRED in OAuth).
- o Replaced examples with hypothetical COSE profile with OSCORE.
- o Added requirement for content type application/ace+cbor in error responses for token and introspection requests and responses.
- o Reworked abbreviation space for claims, request and response parameters.
- o Added text that the RS may indicate that it is busy at the authz-info resource.
- o Added section that specifies how the RS verifies an access token.
- o Added section on the protection of the authz-info endpoint.
- o Removed the expiration mechanism based on sequence numbers.
- o Added reference to [RFC7662](#) security considerations.
- o Added considerations on minimal security requirements for communication.
- o Added security considerations on unprotected information sent to authz-info and in the error responses.

#### **F.6. Version -15 to -16**

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

#### **F.7. Version -14 to -15**

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



- o Editorial improvements.

#### **F.8. Version -13 to -14**

- o Split out the 'aud', 'cnf' and 'rs\_cnf' parameters to [\[I-D.ietf-ace-oauth-params\]](#)
- o Introduced the "application/ace+cbor" Content-Type.
- o Added claim registrations from 'profile' and 'rs\_cnf'.
- o Added note on schema part of AS Information [Section 5.1.2](#)
- o Realigned the parameter abbreviations to push rarely used ones to the 2-byte encoding size of CBOR integers.

#### **F.9. Version -12 to -13**

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

#### **F.10. Version -11 to -12**

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

#### **F.11. Version -10 to -11**

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

#### **F.12. Version -09 to -10**

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



**[F.13.](#) Version -08 to -09**

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

**[F.14.](#) Version -07 to -08**

- o Moved AS discovery from the DTLS profile to the framework, see [Section 5.1](#).
- o Made the use of CBOR mandatory. If you use JSON you can use vanilla OAuth.
- o Made it mandatory for profiles to specify C-AS security and RS-AS security (the latter only if introspection is supported).
- o Made the use of CBOR abbreviations mandatory.
- o Added text to clarify the use of token references as an alternative to CWTs.
- o Added text to clarify that introspection must not be delayed, in case the RS has to return a client token.
- o Added security considerations about leakage through unprotected AS discovery information, combining profiles and leakage through error responses.
- o Added privacy considerations about leakage through unprotected AS discovery.
- o Added text that clarifies that introspection is optional.
- o Made profile parameter optional since it can be implicit.
- o Clarified that CoAP is not mandatory and other protocols can be used.
- o Clarified the design justification for specific features of the framework in [appendix A](#).
- o Clarified [appendix E.2](#).
- o Removed specification of the "cnf" claim for CBOR/COSE, and replaced with references to [[I-D.ietf-ace-cwt-proof-of-possession](#)]

**[F.15.](#) Version -06 to -07**

- o Various clarifications added.
- o Fixed erroneous author email.





**[F.16.](#) Version -05 to -06**

- o Moved sections that define the ACE framework into a subsection of the framework [Section 5](#).
- o Split section on client credentials and grant into two separate sections, [Section 5.2](#), and [Section 5.3](#).
- o Added [Section 5.4](#) on AS authentication.
- o Added [Section 5.5](#) on the Authorization endpoint.

**[F.17.](#) Version -04 to -05**

- o Added [RFC 2119](#) language to the specification of the required behavior of profile specifications.
- o Added [Section 5.3](#) on the relation to the OAuth2 grant types.
- o Added CBOR abbreviations for error and the error codes defined in OAuth2.
- o Added clarification about token expiration and long-running requests in [Section 5.8.3](#)
- o Added security considerations about tokens with symmetric pop keys valid for more than one RS.
- o Added privacy considerations section.
- o Added IANA registry mapping the confirmation types from [RFC 7800](#) to equivalent COSE types.
- o Added [appendix D](#), describing assumptions about what the AS knows about the client and the RS.

**[F.18.](#) Version -03 to -04**

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

**[F.19.](#) Version -02 to -03**

- o Removed references to [draft-ietf-oauth-pop-key-distribution](#) since the status of this draft is unclear.
- o Copied and adapted security considerations from [draft-ietf-oauth-pop-key-distribution](#).
- o Renamed "client information" to "RS information" since it is information about the RS.
- o Clarified the requirements on profiles of this framework.
- o Clarified the token endpoint protocol and removed negotiation of "profile" and "alg" ([section 6](#)).
- o Renumbered the abbreviations for claims and parameters to get a consistent numbering across different endpoints.
- o Clarified the introspection endpoint.



- o Renamed token, introspection and authz-info to "endpoint" instead of "resource" to mirror the OAuth 2.0 terminology.
- o Updated the examples in the appendices.

#### **F.20. Version -01 to -02**

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

#### **F.21. Version -00 to -01**

- o Changed 5.1. from "Communication Security Protocol" to "Client Information".
- o Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP access token request profile for IoT.
  - \* Allow the client to indicate preferences for the communication security protocol.
  - \* Defined the term "Client Information" for the additional information returned to the client in addition to the access token.
  - \* Require that the messages between AS and client are secured, either with (D)TLS or with COSE\_Encrypted wrappers.
  - \* Removed dependency on OSCOAP and added generic text about object security instead.
  - \* Defined the "rpk" parameter in the client information to transmit the raw public key of the RS from AS to client.
  - \* (D)TLS MUST use the PoP key in the handshake (either as PSK or as client RPK with client authentication).
  - \* Defined the use of x5c, x5t and x5tS256 parameters when a client certificate is used for proof of possession.
  - \* Defined "tktn" parameter for signaling for how to transfer the access token.
- o Added 5.2. the CoAP Access-Token option for transferring access tokens in messages that do not have payload.
- o 5.3.2. Defined success and error responses from the RS when receiving an access token.



- o 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
- o [Appendix B](#) Added list of roles and responsibilities for C, AS and RS.

#### Authors' Addresses

Ludwig Seitz  
RISE  
Scheelewaegen 17  
Lund 223 70  
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander  
Ericsson  
Faroegatan 6  
Kista 164 80  
Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem  
Sweden

Email: erik@wahlstromstekniska.se

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

Email: erdtman@spotify.com

Hannes Tschofenig  
Arm Ltd.  
Absam 6067  
Austria

Email: Hannes.Tschofenig@arm.com

