ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

L. Seitz
SICS
G. Selander
Ericsson
E. Wahlstroem
S. Erdtman
Nexus Technology
H. Tschofenig
ARM Ltd.
October 19, 2015

Authorization for the Internet of Things using OAuth 2.0
draft-seitz-ace-oauth-authz-00

Abstract

This memo defines how to use OAuth 2.0 as an authorization framework with Internet of Things (IoT) deployments, thus bringing a well-known and widely used security solution to IoT devices. Where possible vanilla OAuth 2.0 is used, but where the limitations of IoT devices require it, profiles and extensions are provided.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Authorization is the process of deciding what an entity ought to be allowed to do. Managing authorization information for a large number of devices and users is often a complex task where dedicated servers are used.

Managing authorization of users, services and their devices with the help of dedicated authorization servers (AS) is a common task, found in enterprise networks as well as on the Web. In its simplest form the authorization task can be described as granting access to a resource hosted on a device, the resource server (RS). This exchange is mediated by one or multiple authorization servers.

We envision that end consumers and enterprises will want to manage their Internet of Things (IoT) devices in the same style and this desire will increase with the number of devices that need to be managed and controlled. The IoT devices may be constrained in various ways including processing, memory, code, energy, etc., as defined 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-connected devices which are not constrained in terms of power, and all levels in between. Thus IoT devices are very different in terms of available processing and message exchange capabilities.

This memo describes how to re-use OAuth 2.0 [RFC6749] to extend authorization to Internet of Things devices with different kinds of constrainedness. At the time of writing OAuth 2.0 is already used with certain types of IoT devices and this document will provide implementers additional guidance for using it in a secure and privacy-friendly way. Where possible the basic OAuth 2.0 mechanisms are used; in some circumstances profiles are defined, for example to support lower the over-the-wire message size and smaller code size.

2. Terminology

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

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

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

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors], such as client (C), resource server (RS), and authorization server (AS). OAuth 2.0 uses the term "endpoint" to denote HTTP resources such as /token and /authorize at the AS, but we will use the term "resource" in this memo to avoid confusion with the CoAP [RFC7252] term "endpoint".

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

3. Overview

This specification describes a framework for authorization in the Internet of Things consisting 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. When CoAP is used over UDP, transport layer security is provided by DTLS 1.2 [RFC6347] instead of TLS 1.2 [RFC5246].

A third building block is CBOR [RFC7049] for encodings where JSON [RFC7159] is not sufficiently compact. CBOR is a binary encoding designed for extremely small code size and fairly small message size. OAuth 2.0 allows access tokens to use different encodings and this document defines such an alternative encoding. The COSE message format [I-D.ietf-cose-msg] is also based on CBOR.

A fourth building block is application layer security, which is used where transport layer security is insufficient. At the time of writing the preferred approach for securing CoAP at the application layer is via the use of COSE [I-D.ietf-cose-msg], which adds object security to CBOR-encoded data. More details about applying COSE to CoAP can be found in OSCOAP [I-D.selander-ace-object-security].

With the building blocks listed above, solutions satisfying various IoT device and network constraints are possible. A list of constraints is described in detail in RFC 7228 [RFC7228] and a description of how the building blocks mentioned above relate to the various constraints can be found in Appendix A.

Luckily, not every IoT device suffers from all constraints. The described framework does, however, takes all these aspects into account and allows several different deployment variants to co-exist rather than mandating a one-size-fits-all solution. We believe this is important to cover the wide range of possible interworking use cases and the different requirements from a security point of view. Once IoT deployments mature, popular deployment variants will be documented in form of profiles.

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

3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain limited access to a resource with the permission of a resource owner. Authorization related information 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 memo:

Access Tokens:

Access tokens are credentials used to access protected resources. An access token is a data structure representing authorization permissions issued to the client. Access tokens are generated by the authorization server and consumed by the resource server. The access token is opaque to the client.

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

Proof of Possession Tokens:

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

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 included in the access token. When this memo uses the term "access token" it is assumed to be a PoP token unless specifically stated otherwise.

The key bound to the access token (aka PoP key) may be based on symmetric as well as on asymmetrical cryptography. The appropriate choice of security depends on the constraints of the IoT devices as well as on the security requirements of the use case.

Symmetric PoP key:

The AS generates a random symmetric PoP key, encrypts it for the RS and includes it inside an 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 then included inside the access token and sent back to the requesting client.

The access token is protected against modifications using a MAC or a digital signature of the AS. The choice of PoP key does not necessarily imply a specific credential type for the integrity protection of the token. More information about PoP tokens can be found in [I-D.ietf-oauth-pop-architecture].

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 request. In turn, the AS may use the "scope" response parameter to inform the client of the scope of the access token issued. This memo uses CBOR encoded messages defined in Appendix C to request scopes and to be informed what scopes the access token was actually authorized for by the AS.

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

Claims:

The information carried in the access token in the form of type-value pairs is called claims. An access token may for example include a claim about the AS that issued the token (the "iss" claim) and what audience the access token is intended for (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 the 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 Appendix D we define a CBOR version of JWT that we call CBOR Web Token (CWT).

Introspection:

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

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 HTTP uses headers and query-strings to convey additional information about a request, CoAP encodes such information in so-called ’options’.

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [I-D.ietf-core-block]. However, this method does not allow the fragmentation of large CoAP options, therefore data encoded in options has to be kept small.

3.3. Object Security

Transport layer security is not always sufficient and application layer security has to be provided. COSE [I-D.ietf-cose-msg] defines a message format for cryptographic protection of data using CBOR encoding. There are two main approaches for application layer security:

Object Security of CoAP (OSCOAP)

> OSCOAP [I-D.selander-ace-object-security] is a method for protecting CoAP request/response message exchanges, including CoAP payloads, CoAP header fields as well as CoAP options. OSCOAP provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages.
>
> A CoAP message protected with OSCOAP contains the CoAP option "Object-Security" which signals that the CoAP message carries a COSE message ([I-D.ietf-cose-msg]). OSCOAP defines a profile of COSE which includes replay protection.

Object Security of Content (OSCON)

> For the case of wrapping of application layer payload data ("content") only, such as resource representations or claims of access tokens, the same COSE profile can be applied to obtain end-to-end confidentiality, integrity and replay protection. [I-D.selander-ace-object-security] defines this functionality as Object Security of Content (OSCON).
>
> In this case, the message is not bound to the underlying application layer protocol and can therefore be used with HTTP, CoAP, Bluetooth Smart, etc. Whereas OSCOAP integrity protects specific CoAP message meta-data like request/response code, and

binds a response to a specific request, since OSCON protects only payload/content, those security features are lost. The advantages are that an OSCON message can be passed across different protocols, from request to response, and used to secure group communications.

4. Protocol Interactions

This framework is based on the same protocol interactions as OAuth 2.0: A client obtains an access token from an AS and presents the token to an RS to gain access to a protected resource. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in Section 3.1.

The consent of the resource owner, for giving a client access to a protected resource, can be pre-configured authorization policies or dynamically at the time when the request is sent. The resource owner and the requesting party (= client owner) are not shown in Figure 1.

For the description in this document we assume that the client has been registered to an AS. Registration means that the two share credentials, configuration parameters and that some form of authorization has taken place. These credentials are used to protect the token request by the client and the transport of access tokens and client information from AS to the client.

It is also assumed that the RS 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 the content cannot be modified, and if needed, that the content is confidentiality protected.

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

At the start of the protocol there is an optional discovery step where the client discovers the resource server and the resources this server hosts. In this step the client might also determine what permissions are needed to access the protected resource. The exact procedure depends on the protocols being used and the specific deployment environment. In Bluetooth Smart, for example, advertisements are broadcasted by a peripheral, including information about the supported services. In CoAP, as a second example, a client can makes a request to "/.well-known/core" to obtain information about available resources, which are returned in a standardized format as described in [RFC6690].

```
     +--------+                                     +---------------+
     |        |---(A)-- Token Request ------------->|               |
     |        |                                     | Authorization |
     |        |<--(B)-- Access Token ---------------|    Server     |
     |        |       + Client Information          |               |
     |        |                                     +---------------+
     |        |                                            ^ |
     |        |        Introspection Request & Response (D)| |(E)
     | Client |                                            | v
     |        |                                     +--------------+
     |        |---(C)-- Token + Request ----------->|              |
     |        |                                     |   Resource   |
     |        |<--(F)-- Protected Resource ---------|    Server    |
     |        |                                     |              |
     +--------+                                     +--------------+
```

Figure 1: Overview of the basic protocol flow

Requesting an Access Token (A):

The client makes an access token request to the AS. This memo assumes the use of PoP tokens (see Section 3.1 for a short description) wherein the AS binds a key to an access token. The client may include permissions it seeks to obtain, and information about the type of credentials it wants to use (i.e., symmetric or asymmetric cryptography).

Access Token Response (B):

If the AS successfully processes the request from the client, it returns an access token. It also includes various parameters, which we call "Client Information". In addition to the response parameters defined by OAuth 2.0 and the PoP token extension, we consider new kinds of response parameters in Section 5, including information on which security protocol the client should use with the resource server(s) that it has just been authorized to access. Communication security between client and RS may be based on pre-provisioned keys/security contexts or dynamically established to the RS via the PoP token; and to the client via the client information as described in Section 5.1.

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, Bluetooth Smart etc., are also possible candidates.

Depending on the device limitations and the selected protocol this exchange may be split up into two phases:

(1) the client sends the access token to a newly defined authorization endpoint at the RS (see Section 5.2) , which conveys authorization information to the RS that may be used for subsequent resource requests, and

(2) the client makes the resource access request, using the communication security protocol and other client information obtained from the AS.

The RS verifies that the token is integrity protected by the AS and compares the claims contained in the access token with the resource request. If the RS is online, validation can be handed over to the AS using token introspection (see messages D and E) over HTTP or CoAP, in which case the different parts of step C may be interleaved with introspection.

Token Introspection Request (D):

A resource server may be configured to use token introspection to interact with the AS to obtain the most recent claims, such as scope, audience, validity etc. associated with a specific access token. Token introspection over CoAP is defined in [I-D.wahlstroem-ace-oauth-introspection] and for HTTP in [I-D.ietf-oauth-introspection].

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 claims associated with it back to the RS. The RS then uses the received claims 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. OAuth 2.0 Profiling

This section describes profiles of OAuth 2.0 adjusting it to constrained environments for use cases where this is necessary.

5.1. Communication Security Protocol

OAuth 2.0 using bearer tokens, as described in RFC 6749 and in RFC 6750, requires TLS for all communication interactions between client, authorization server, and resource server. This is possible in the scope where OAuth 2.0 was originally developed, web and mobile applications. In these environments resources like computational power and bandwidth are not scarce and operating systems as well as browser platforms are pre-provisioned with trust anchors that enable clients to authenticate servers based on the Web PKI. In a more heterogeneous IoT environment a wider range of use cases needs to be supported. Therefore, this document suggests extensions to OAuth 2.0 that enable the AS to inform the client on how to communicate securely with a RS.

The client and the RS might not have any prior knowledge about each other, therefore the AS needs to help them to establish a security context or at least a key. The AS does this by indicating communication security protocol ("csp") and additional key parameters in the client information.

The "csp" parameter specifies how client and RS communication is going to be secured based on returned keys. Currently defined values are "TLS", "DTLS", "OSCOAP" and "OSCON". Depending on the value different additional parameters become mandatory.

TLS with certificates may make use of pre-established trust anchors or configured more tightly with additional client information parameters, like x5c, x5t or x5t#S256.

CoAP specifies three security "modes" of DTLS: PreSharedKey, RawPublicKey and Certificate. In case of PreSharedKey and RawPublicKey DTLS is based on the use keys distributed in the PoP token and via the client information. Additional certificate information may also be added, for example using the parameter x5c, x5t or x5t#S256.

To use OSCOAP and OSCON requires security context to be established, which can be provisioned with PoP token and client information, or derived from keys provisioned in this way.

5.2. Authorization Information Resource at the Resource Server

A consequence of allowing the use of CoAP as web transfer protocol is that we cannot rely on HTTP specific mechanisms, such as transferring information elements in HTTP headers since those are not necessarily gracefully mapped to CoAP. In case the access token is larger then 255 bytes it should not be sent as a CoAP option.

For conveying authorization information to the RS we therefore introduce a new resource to which the PoP tokens can be sent to convey authorization information before the first resource request is made by the client. This specification calls this resource "/authz-info"; the URI may, however, vary in deployments.

5.3. Authorization Information Format

We introduce a new claim for describing access rights with a specific format, the "aif" claim. In this memo we propose to use the compact format provided by AIF [I-D.bormann-core-ace-aif]. Access rights may be specified as a list of URIs of resources together with allowed actions (GET, POST, PUT, PATCH, or DELETE).

5.4. CBOR Data Formats

The /token resource (called "endpoint" in OAuth 2.0), defined in Section 3.2 of [RFC6749], is used by the client to obtain an access token. Requests sent to the /token resource use the HTTP POST method and the payload includes a query component, which is formatted as application/x-www-form-urlencoded. CoAP payloads cannot be formatted in the same way which requires the /token resource on the AS to be profiled. Appendix C defines a CBOR-based format for sending parameters to the /token resource.

5.5. CBOR Web Token

CBOR Web Tokens (CWT) are defined in Appendix D as compact analogs of JSON Web Tokens (JWT) [RFC7519]. CWTs uses COSE [I-D.ietf-cose-msg] to offer similar, but more compact security services. CWT supports PoP token functionality.

6. Deployment Scenarios

There is a large variety of IoT deployments, as is indicated in Appendix A, and this section highlights 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 us to also consider how these requests and responses between the clients and the resource servers are secured.

The security protocols between other pairs of nodes in the architecture, namely client-to-AS and RS-to-AS, are not detailed in these examples. Different security protocols may be used on transport or application layer.

Note: We use the CBOR diagnostic notation for examples of requests and responses.

6.1. Client and Resource Server are Offline

In this scenario we consider the case where both the resource server and the client are offline, i.e., they are not connected to the AS at the time of the resource request. This access procedure involves steps A, B, C, and F of Figure 1, but assumes that step A and B have been carried out during a phase when the client had connectivity to AS.

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 2.

A: The client first generates a public-private key pair used for communication security with the RS.

The client sends the POST request to /token at AS. The request contains the public key of the client and the Audience parameter set to "tempSensorInLivingRoom", a value the that the temperature sensor identifies itself with. The AS evaluates the request and authorizes the client to access the resource.

B: The AS responds with a PoP token and client information. The PoP token contains the public key of the client, while the client information contains the public key of the RS. For communication security this example uses DTLS with raw public keys between the client and the RS.

Note: In this example we assume that the client knows what resource it wants to access, and is therefore able to request specific audience and scope claims for the access token.

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

Figure 2: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 3.

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

                  Response-Payload :
                  {
                    "access_token" : b64’SlAV32hkKG ...’,
                    "token_type" : "pop",
                    "csp" : "DTLS",
                    "key" : b64’eyJhbGciOiJSU0ExXzUi ...’
                  }
```

Figure 3: Request and Response Payload Details.

The content of the "key" parameter and the access token are shown in Figure 4 and Figure 5.

```
        {
          "kid" : b64’c29tZSBwdWJsaWMga2V5IGlk’,
          "kty" : "EC",
          "crv" : "P-256",
          "x"   : b64’MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4’,
          "y"   : b64’4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM’
        }
```

Figure 4: Public Key of the RS.

```
       {
         "aud" : "tempSensorInLivingRoom",
         "iat" : "1360189224",
         "cnf" : {
           "jwk" : {
             "kid" : b64’1Bg8vub9tLe1gHMzV76e8’,
             "kty" : "EC",
             "crv" : "P-256",
             "x" : b64’f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU’,
             "y" : b64’x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0’
           }
         }
       }
```

Figure 5: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 6 - Figure 7.

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

The client and resource server run 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.

F: The RS responds with a resource representation over DTLS.

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

Figure 6: Access Token provisioning to RS

```
                      Resource
            Client     Server
               |         |
               |<=======>| DTLS Connection Establishment
               |         |   using Raw Public Keys
               |         |
               |         |
               +-------->| Header: GET (Code=0.01)
               | GET     | Uri-Path: "temperature"
               |         |
               |         |
               |         |
           F:  |<--------+ Header: 2.05 Content
               | 2.05    | Payload: {"t":"22.7"}
               |         |
```

Figure 7: Resource Request and Response protected by DTLS.

6.2. Resource Server Offline

In this deployment scenario we consider the case of an RS that may not be able to access the AS at the time it receives an access request from a client. We denote this case "RS offline", it involves steps A, B, C and F of Figure 1.

If the RS is offline, then it must be possible for the RS to locally validate the access token. This requires self-contained tokens to be used.

The validity time for the token should always be chosen as short as possible to reduce the possibility that a token contains out-of-date authorization information. Therefore the value for the Expiration Time claim ("exp") should be set only slightly larger than the value for the Issuing Time claim ("iss"). A constrained RS with means to reliably measure time must validate the expiration time of the access token.

The following example shows interactions between a client (AC control unit), an offline resource server (temperature sensor) and an authorization server. The message exchanges A and B are shown in Figure 8.

A: The client sends the request POST to /token at AS. The request contains the Audience parameter set to "tempSensor109797", a value that the temperature sensor identifies itself with. The scope the client want’s the AS to authorize the access token for is "owner", which means that the token can be used to both read temperature data and upgrade the firmware on the RS. The AS evaluates the request and authorizes the client to access the resource.

B: The AS responds with a PoP token and client information. The PoP token is wrapped in a COSE message, object secured content from AS to RS. The client information contains a symmetric key. In this case communication security between C and RS is OSCOAP with an authenticated encryption algorithm. The client derives two unidirectional security contexts to use with the resource request and response messages. The access token includes the claim "aif" with the authorized access that an owner of the temperature device can enjoy. The "aif" claim, issued by the AS, informs the RS that the owner of the access token, that can prove the possession of a key is authorized to make a GET request against the /tempC resource and a POST request on the /firmware resource.

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

Figure 8: Token Request and Response

The information contained in the Request-Payload and the Response-Payload is shown in Figure 9.

```
                  Request-Payload:
                  {
                    "grant_type" : "client_credentials",
                    "client_id" : "myclient",
                    "client_secret" : "qwerty",
                    "aud" : "tempSensor109797",
                    "scope" : "owner"
                  }

                  Response-Payload:
                  {
                    "access_token": b64'SlAV32hkKG ...',
                    "token_type" : "pop",
                    "csp" : "OSCOAP",
                    "key" : b64'eyJhbGciOiJSU0ExXzUi ...'
                  }
```

Figure 9: Request and Response Payload for RS offline

Figure 10 shows examples of the key and the access_token parameters of the Response-Payload, decoded to CBOR.

```
access_token:
{
  "aud" : "tempSensor109797",
  "exp" : 1311281970,
  "iat" : 1311280970,
  "aif" :  [["/tempC", 0], ["/firmware", 2]],
  "cnf" : {
    "ck":b64'JDLUhTMjU2IiwiY3R5Ijoi ...'
    }
 }

key:
{
  "alg" : "AES_128_CCM_8",
  "kid" : b64'U29tZSBLZXkgSWQ',
  "k" : b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
}
```

Figure 10: Access Token and symmetric key from the Response-Payload

Message exchanges C and F are shown in Figure 11 and Figure 12.

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

The client sends the CoAP requests GET to /tempC on the RS using OSCOAP. The RS verifies the request and that it is authorized.

F: The RS responds with a protected status code using OSCOAP. The client verifies the response.

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

Figure 11: Access Token provisioning to RS

```
                     Resource
           Client     Server
             |          |
             +--------->| Header: GET (Code=0.01)
             | GET      | Object-Security:
             |          |   (<seq>,<cid>,[Uri-Path:"tempC"],<tag>)
             |          |
         F:  |<---------+ Header: 2.05 Content
             | 2.05     | Object-Security:
             |          |   (<seq>,<cid>,[22.7 C],<tag>)
             |          |
```

Figure 12: Resource request and response protected by OSCOAP

In Figure 12 the GET request contains an Object-Security option and an indication of the content of the COSE object: a sequence number ("seq", starting from 0), a context identifier ("cid") indicating the security context, the ciphertext containing the encrypted CoAP option identifying the resource, and the Message Authentication Code ("tag") which also covers the Code in the CoAP header.

The Object-Security ciphertext in the response [22.7 C] represents an encrypted temperature reading. (The COSE object is actually carried in the CoAP payload when possible but that is omitted to simplify notation.)

6.3. Token Introspection with an Offline Client

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

Since the client is assumed to be offline, at least for a certain period of time, a pre-provisioned access token has to be long-lived. The resource server may use its online connectivity to validate the access token with the authorization server, which is shown in the example below.

In the example we show the interactions between an offline client (key fob), a resource server (online lock), and an authorization server. We assume that there is a provisioning step where the client has access to the AS. This corresponds to message exchanges A and B which are shown in Figure 13.

A: The client sends the request using POST to /token at AS. The request contains the Audience parameter set to "lockOfDoor4711", a value the that the online door in question identifies itself with. The AS generates an access token as on opaque string, which it can match to the specific client, a targeted audience and a symmetric key security context.

B: The AS responds with the an access token and client information, the latter containing a symmetric key. Communication security between C and RS will be OSCOAP with authenticated encryption.

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

Figure 13: Token Request and Response using Client Credentials.

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 is used to authorize the key for his car at the the car manufacturers AS.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 14.

```
               Request-Payload:
               {
                 "grant_type" : "token",
                 "aud" : "lockOfDoor4711",
                 "client_id" : "myclient",
               }

               Response-Payload:
               {
                 "access_token" : b64'SlAV32hkKG ...'
                 "token_type" : "pop",
                 "csp" : "OSCOAP",
                 "key" : b64'eyJhbGciOiJSU0ExXzUi ...'
               }
```

Figure 14: Request and Response Payload for C offline

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

C: Next, the client POSTs the access token to the /authz-info resource in the RS. This is a plain CoAP request, i.e. no DTLS/ OSCOAP 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 step E and only acknowledges on the CoAP message layer (indicated with a dashed line).

```
                      Resource
             Client     Server
               |          |
           C:  +--------->| Header: POST (T=CON, Code=0.02
               | POST     | Token 0x2a12)
               |          | Uri-Path:"authz-info"
               |          | Payload: SlAV32hkKG ...
               |          |   (access token)
```

```
                  |         |
                  |<- - - - + Header: T=ACK
                  |         |
```

Figure 15: Access Token provisioning to RS

D: The RS forwards the token to the /introspect resource on the AS. Introspection assumes a secure connection between the AS and the RS, e.g. using DTLS or OSCOAP, which is not detailed in this example.

E: The AS provides the introspection response containing claims about the token. This includes the confirmation key (cnf) claim 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 Code 2.04 (Changed), using CoAP Token 0x2a12. This step is not shown in the figures.

```
         Resource Authorization
          Server     Server
             |          |
         D:  +--------->| Header: POST (Code=0.02)
             |  POST    | Uri-Path: "introspect"
             |          | Payload: <Request-Payload>
             |          |
         E:  |<---------+ Header: 2.05 Content
             |  2.05    | Content-Type: application/cbor)
             |          | Payload: <Response-Payload>
             |          |
```

Figure 16: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 17.

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

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

Figure 17: Request and Response Payload for Introspection

The client sends the CoAP requests PUT 1 (= "close the lock") to / lock on RS using OSCOAP with a security context derived from the key supplied in step B. The RS verifies the request with the key supplied in step E and that it is authorized by the token supplied in step C.

F: The RS responds with a protected status code using OSCOAP. The client verifies the response.

```
                   Resource
          Client    Server
            |         |
            +-------->| Header: PUT (Code=0.03)
            | PUT     | Object-Security:
            |         |    (<seq>,<cid>,[Uri-Path:"lock", 1],<tag>)
            |         |
        F:  |<--------+ Header: 2.04 Changed
            | 2.04    | Object-Security:
            |         |    (<seq>,<cid>,,<tag>)
            |         |
```

Figure 18: Resource request and response protected by OSCOAP

The Object-Security ciphertext [...] of the PUT request contains CoAP options that are encrypted, as well as the payload value ’1’ which is the value of PUT to the door lock.

In this example there is no ciphertext of the PUT response, but "tag" contains a MAC which covers the request sequence number and context identifier as well as the Code which allows the Client to verify that this actuator command was well received (door is locked).

6.4. Always-On Connectivity

A popular deployment scenario for IoT devices is to have them always be connected to the Internet so that they can be reachable to receive commands. As a continuation from the previous scenarios we assume that both the client and the RS are online at the time of the access request.

If the client and the resource server are online then the AS should be configured to issue short-lived access tokens for the resource to the client. The resource server must then validate self-contained access tokens or otherwise must use token introspection to obtain the up-to-date claim information. If transmission costs are high or the channel is lossy, the CWT token format may be used instead of a JWT to reduce the volume of network traffic. In terms of messaging this deployment scenario uses the patterns described in the previous sub-sections.

Note that despite the lack of connectivity constraints there may still be other restrictions a deployment may face.

6.5. Token-less Authorization

In this deployment scenario we consider the case of an RS which is severely energy constrained, sleeps most of the time and need to have a tight messaging budget. It is not only infeasible to access the AS at the time of the access request, as in the "RS offline" case Section 6.2, it must be offloaded as much message communication as possible.

OAuth 2.0 is already an efficient protocol in terms of message exchanges and can be further optimized by compact encodings of tokens. The scenario illustrated in this section goes beyond that and removes the access tokens from the protocol. This may be considered a degenerate case of OAuth 2.0 but it allows us to do two things:

1. The common case where authorization is performed by means of authentication fits into the same protocol framework. Authentication protocol and key is specified by client information, and access token is omitted.

2. Authentication, and thereby authorization, may even be implicit, i.e. anyone with access to the right key is authorized to access the protected resource.

In case 2., the RS does not need to receive any message from the client, and therefore enables offloading recurring resource request and response processing to a third party, such as a Message Broker (MB) in a publish-subscribe setting.

This scenario involves steps A, B, C and F of Figure 1 and four parties: a client (subscriber), an offline RS (publisher), a trusted AS, and a MB, not necessarily trusted with access to the plain text publications. Message exchange A, B is shown in Figure 19.

A: The client sends the request POST to /token at AS. The request contains the Audience parameter set to "birchPollenSensor301", a value that characterizes a certain pollen sensor resource. The AS evaluates the request and authorizes the client to access the resource.

B: The AS responds with an empty token and client information with a security context to be used by the client. The empty token signifies that authorization is performed by means of authentication using the communication security protocol indicated with "csp". In this case it is object security of content (OSCON) i.e. protection of CoAP payload only. The security context contains the symmetric decryption key and a public signature verification key of the RS.

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

Figure 19: Token Request and Response

The information contained in the Request-Payload and the Response-Payload is shown in Figure 20.

```
               Request-Payload :
               {
                 "grant_type" : "client_credentials",
                 "aud" : "birchPollenSensor301",
                 "client_id" : "myclient",
                 "client_secret" : "qwerty"
               }

               Response-Payload :
               {
                 "access_token" : NULL,
                 "token_type" : "none",
                 "csp" : "OSCON",
                 "key" : b64’eyJhbGciOiJSU0ExXzUi ...’
               }
```

Figure 20: Request and Response Payload for RS severely constrained

The content of the "key" parameter is shown in Figure 21.

```
        key :
        {
          "alg" : "AES_128_CTR_ECDSA",
          "kid" : b64’c29tZSBvdGhlciBrZXkgaWQ’;
          "k"   : b64’ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE’,
          "crv" : "P-256",
          "x"   : b64’MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4’,
          "y"   : b64’4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM’
        }
```

Figure 21: The ’key’ Parameter

The RS, which sleeps most of the time, occasionally wakes up, measures the number birch pollens per cubic meters, publishes the measurements to the MB, and then returns to sleep. See Figure 22.

In this case the birch pollen count stopped at 270, which is encrypted with the symmetric key and signed with the private key of the RS. The MB verifies that the message originates from RS using the public key of RS, that it is not a replay of an old measurement using the sequence number of the OSCON COSE profile, and caches the object secured content. The MB does not have the secret key so is unable to read the plain text measurement.

Message exchanges C and F are shown in Figure 22.

C: Since there is no access token, the client does not address the /authz-info resource in the RS. The client sends the CoAP request GET to /birchPollen on MB which is a plain CoAP request.

F: The MB responds with the cached object secured content.

```
                 Message   Resource
        Client    Broker    Server
          |         |         |
          |         |<--------| Header: PUT (Code=0.02)
          |         |  PUT    | Uri-Path: "birchPollen"
          |         |         | Payload: (<seq>,<cid>,["270"],<tag>)
          |         |         |
          |         |-------->| Header: 2.04 Changed
          |         | 2.04    |
          |         |
          |         |
      C:  +-------->| Header: GET (Code=0.01)
          | GET     | Uri-Path: "birchPollen"
          |         |
          |         |
      F:  |<--------+ Header: 2.05 Content
          | 2.05    | Payload: (<seq>,<cid>,["270"],<tag>)
          |         |
```

Figure 22: Sensor measurement protected by COSE

The payload is a COSE message consisting of sequence number ’seq’ stepped by the RS for each publication, the context identifier ’cid’ in this case coinciding with the key identifier ’kid’ of Figure 21, the encrypted measurement and the signature by the RS.

Note that the same COSE message format may be used as in OSCOAP but that only CoAP payload is protected in this case.

The authorization step is implicit, so while any client could request access the COSE object, only authorized clients have access to the symmetric key needed to decrypt the content.

Note that in this case the order of the message exchanges A,B and C,F could in principle be interchanged, i.e. the client could first request and obtain the protected resource in steps C,F; and after that request client information containing the keys decrypt and verify the message.

6.6. Securing Group Communication

There are use cases that require securing communication between a (group of) senders and a group of receivers. One prominent example is lighting. Often, a set of lighting nodes (e.g., luminaires, wall-switches, sensors) are grouped together and only authorized members of the group must be able read and process messages. Additionally, receivers of group messages must be able to verify the integrity of received messages as being generated within the group.

The requirements for securely communicating in such group use cases efficiently is outlined in [I-D.somaraju-ace-multicast] along with an architectural description that aligns with the content of this document. The requirements for conveying the necessary identifiers to reference groups and also the process of commissioning devices can be accomplished using the protocol described in this document. For details about the lighting-unique use case aspects, the architecture, as well as other multicast-specific considerations we refer the reader to [I-D.somaraju-ace-multicast].

7. Security Considerations

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work, as well as the security considerations from [I-D.ietf-ace-actors]. Furthermore [RFC6819] provides additional security considerations for OAuth which apply to IoT deployments as well. Finally [I-D.ietf-oauth-pop-architecture] discusses security and privacy threats as well as mitigation measures for Proof-of-Possession tokens.

8. IANA Considerations

TBD

9. Acknowledgments

We would like to thank Eve Maler for her contributions to the use of OAuth 2.0 and UMA in IoT scenarios, Robert Taylor for his discussion input, and Malisa Vucinic for his input on the ACRE proposal FIXME:REF which was one source of inspiration for this work. Finally, we would like to thank the ACE working group in general for their feedback.

10. References

10.1. Normative References

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-02 (work in progress), March 2015.

[I-D.ietf-cose-msg]
Schaad, J. and B. Campbell, "CBOR Encoded Message Syntax", draft-ietf-cose-msg-05 (work in progress), September 2015.

[I-D.ietf-oauth-introspection]
Richer, J., "OAuth 2.0 Token Introspection", draft-ietf-oauth-introspection-09 (work in progress), May 2015.

[I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-02 (work in progress), July 2015.

[I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-01 (work in progress), March 2015.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., and L. Seitz, "March 9, 2015", draft-selander-ace-object-security-01 (work in progress), March 2015.

[I-D.wahlstroem-ace-oauth-introspection]
Wahlstroem, E., "OAuth 2.0 Introspection over the Constrained Application Protocol (CoAP)", draft-wahlstroem-ace-oauth-introspection-01 (work in progress), March 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

10.2. Informative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-00 (work in progress), August 2015.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., and H. Tschofenig, "Multicast Security for the Lighting Domain", draft-somaraju-ace-multicast-00 (work in progress), July 2015.

[RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, October 2006.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

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

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

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

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

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

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

Appendix A. Design Justification

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

Common IoT constraints are:

Low Power Radio:

Many IoT devices are equipped with a small battery which needs to last for a long time. For many constrained wireless devices the highest energy cost is associated to transmitting or receiving messages. It is therefore important to keep the total communication overhead low, including minimizing the number and size of messages sent and received, which has an impact of choice of 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 can be optimized. For example authentication and key establishment may in certain cases where security requirements so allows be replaced by provisioning of security context by a trusted third party, using transport or application layer security.

Low CPU Speed:

Some IoT devices are equipped with processors that are significantly slower than those found in most current devices on the Internet. This typically has implications on what timely cryptographic operations a device is capable to perform, which in turn impacts e.g. protocol latency. Symmetric key cryptography may be used instead of the computationally more expensive public key cryptography where the security requirements so allows, but this may also require support for trusted third party assisted secret key establishment using transport or application layer security.

Small Amount of Memory:

Microcontrollers embedded in IoT devices are often equipped with small amount of RAM and flash memory, which places limitations what kind of processing can be performed and how much code can be put on those devices. To reduce code size fewer and smaller protocol implementations can be put on the firmware of such a device. In this case, CoAP may be used instead of HTTP, symmetric key cryptography instead of public key cryptography, and CBOR instead of JSON. Authentication and key establishment protocol, e.g. the DTLS handshake, in comparison with assisted key establishment also has an impact on memory and code.

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers do not want to give access to the data collected by their IoT device or to functions it may offer to third parties. Since the classical approach of requesting permissions from end users via a rich user interface does not work in many IoT deployment scenarios these functions need to be delegated to user controlled devices that are better suitable for such tasks, such as smart phones and tablets.

Communication Constraints:

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

The communication interactions this framework builds upon (as shown graphically in Figure 1) may be accomplished using a variety of different protocols, and not all parts of the message flow are used in all applications due to the communication constraints. While we envision deployments to make use of CoAP we explicitly want to support HTTP, HTTP/2 or specific protocols, such as Bluetooth Smart communication, which does not necessarily use IP. The latter raises the need for application layer security over the various interfaces.

Appendix B. Optimizations

This section sketches some potential optimizations to the presented solution.

Access token in DTLS handshake

In the case of CSP=DTLS/TLS, the access token provisoning exchange in step C of the protocol may be embedded in the security handshake. Different solutions are possible, where one standardized method would be the use of the TLS supplemental data extension [RFC4680] for transferring the access token.

Reference token and introspection

In case of introspection it may be useful with access tokens which are not self-contained (also known as "reference tokens") that are used to lookup detailed information about the authorization. The RS uses the introspection message exchange not only for validating token claims, but also for obtaining claims that potentially were not known at the time when the access token was issued.

A reference token can be made much more compact than a CWT, since it does not need to contain any of claims that it represents. This could be very useful in particular if the client is constrained and offline most of the time.

Reference token in CoAP option

While large access tokens must be sent in CoAP payload, if the access token is known to be of a certain limited size, for example in the case of a reference token, then it would be favorable to combine the access token provisioning request with the resource request to the RS.

One way to achieve this is to define a new CoAP option for carrying reference tokens, called "Ref-Token" as shown in the example in Figure 23.

```
                   Resource
          Client    Server
            |         |
        C:  +-------->| Header: PUT (Code=0.02)
            | PUT     | Ref-Token:SlAV32hkKG
            |         | Object-Security:
            |         |    <seq>,<cid>,[Uri-Path:"lock", 1],<tag>)
            |         |
            .         .
            .         .
            .         .
            |         |
        F:  |<--------+ Header: 2.04 Changed
            | 2.04    | Object-Security:
            |         |    (<seq>,<cid>,,<tag>)
            |         |
```

Figure 23: Reference Token in CoAP Option

Appendix C. CoAP and CBOR profiles for OAuth 2.0

Many IoT devices can support OAuth 2.0 without any additional extensions, but for certain constrained settings additional profiling is needed. In this appendix we define CoAP resources for the HTTP based token and introspection endpoints used in vanilla OAuth 2.0. We also define a CBOR alternative to the JSON and form based POST structures used in HTTP.

C.1. Profile for Token resource

The token resource is used by the client to obtain an access token by presenting its authorization grant or client credentials to the / token resource the AS.

C.1.1. Token Request

The client makes a request to the token resource by sending a CBOR structure with the following attributes.

grant_type:

REQUIRED. The grant type, "code", "client_credentials", "password" or others.

client_id:

OPTIONAL. The client identifier issued to the holder of the token (client or RS) during the registration process.

client_secret:

OPTIONAL. The client secret.

scope:

OPTIONAL. The scope of the access request as described by Section 3.1.

aud:

OPTIONAL. Service-specific string identifier or list of string identifiers representing the intended audience for this token, as defined in CWT Appendix D.

alg:

OPTIONAL. The value in the ’alg’ parameter together with value from the ’token_type’ parameter allow the client to indicate the supported algorithms for a given token type.

key:

OPTIONAL. This field contains information about the public key the client would like to bind to the access token in the COSE Key Structure format.

The parameters defined above use the following CBOR major types.

Value	Major Type	Key
0	0	grant_type
1	0	client_id
2	0	client_secret
3	0	scope
4	0	aud
5	0	alg
6	0	key

Figure 24: CBOR mappings used in token requests

C.1.2. Token Response

The AS responds by sending a CBOR structure with the following attributes.

access_token:

REQUIRED. The access token issued by the authorization server.

token_type:

REQUIRED. The type of the token issued. "pop" is recommended.

key:

REQUIRED, if symmetric key cryptography is used. A COSE Key Structure containing the symmetric proof of possession key. The members of the structure can be found in section 7.1 of [I-D.ietf-cose-msg].

csp:

REQUIRED. Information on what communication protocol to use in the communication between the client and the RS. Details on possible values can be found in Section 5.1.

scope:

OPTIONAL, if identical to the scope requested by the client; otherwise, REQUIRED.

alg:

OPTIONAL. The ’alg’ parameter provides further information about the algorithm, such as whether a symmetric or an asymmetric crypto-system is used.

The parameters defined above use the following CBOR major types.

Value	Major Type	Key
0	0	access_token
1	0	token_type
2	0	key
3	0	csp
4	0	scope
5	0	alg

Figure 25: CBOR mappings used in token responses

C.2. CoAP Profile for OAuth Introspection

This section defines a way for a holder of access tokens, mainly clients and RS’s, to get metadata like validity status, claims and scopes found in access token. The OAuth Token Introspection specification [I-D.ietf-oauth-introspection] defines a way to validate the token using HTTP POST or HTTP GET. This document reuses the work done in the OAuth Token Introspection and defines a mapping of the request and response to CoAP [RFC7252] to be used by constrained devices.

C.2.1. Introspection Request

The token holder makes a request to the Introspection CoAP resource by sending a CBOR structure with the following attributes.

token:

REQUIRED. The string value of the token.

resource_id:

OPTIONAL. A service-specific string identifying the resource that the client doing the introspection is asking about.

client_id:

OPTIONAL. The client identifier issued to the holder of the token (client or RS) during the registration process.

client_secret:

OPTIONAL. The client secret.

The parameters defined above use the following CBOR major types:

Value	Major Type	Key
0	0	token
1	0	resource_id
2	0	client_id
3	0	client_secret

Figure 26: CBOR Mappings to Token Introspection Request Parameters.

C.2.2. Introspection Response

If the introspection request is valid and authorized, the authorization server returns a CoAP message with the response encoded as a CBOR structure in the payload of the message. If the request failed client authentication or is invalid, the authorization server returns an error response using the CoAP 4.00 ’Bad Request’ response code.

The JSON structure in the payload response includes the top-level members defined in Section 2.2 in the OAuth Token Introspection specification [I-D.ietf-oauth-introspection]. It is RECOMMENDED to only return the ’active’ attribute considering constrained nature of CoAP client and server networks.

Introspection responses in CBOR use the following mappings:

active:

REQUIRED. The active key is an indicator of whether or not the presented token is currently active. The specifics of a token’s "active" state will vary depending on the implementation of the authorization server, and the information it keeps about its tokens, but a "true" value return for the "active" property will generally indicate that a given token has been issued by this authorization server, has not been revoked by the resource owner, and is within its given time window of validity (e.g., after its issuance time and before its expiration time).

scope:

OPTIONAL. A string containing a space-separated list of scopes associated with this token, in the format described in Section 3.3 of OAuth 2.0 [RFC6749].

client_id:

OPTIONAL. Client identifier for the client that requested this token.

username:

OPTIONAL. Human-readable identifier for the resource owner who authorized this token.

token_type:

OPTIONAL. Type of the token as defined in Section 5.1 of OAuth 2.0 [RFC6749] or PoP token.

exp:

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire, as defined in CWT Appendix D.

iat:

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire, as defined in CWT Appendix D.

nbf:

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire, as defined in CWT Appendix D.

sub:

OPTIONAL. Subject of the token, as defined in CWT Appendix D. Usually a machine-readable identifier of the resource owner who authorized this token.

aud:

OPTIONAL. Service-specific string identifier or list of string identifiers representing the intended audience for this token, as defined in CWT Appendix D.

iss:

OPTIONAL. String representing the issuer of this token, as defined in CWT Appendix D.

cti:

OPTIONAL. String identifier for the token, as defined in CWT Appendix D

The parameters defined above use the following CBOR major types:

Value	Major Type	Key

0	0	active
1	0	scopes
2	0	client_id
3	0	username
4	0	token_type
5	0	exp
6	0	iat
7	0	nbf
8	0	sub
9	0	aud
10	0	iss
11	0	cti

Figure 27: CBOR Mappings to Token Introspection Response Parameters.

Appendix D. CBOR Web Token (CWT)

CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. CWT is a profile of JSON Web Tokens that is optimized for constrained devices. The claims in a CWT are encoded in CBOR and COSE is used for signature and encryption. A claim is a piece of information asserted about a subject. A claim is represented as a name/value pair consisting of a Claim Name and a Claim Value.

The suggested pronunciation of CWT is the same as the English word "cot".

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

D.1. Claim Names

The following Claim Names are asserted by the AS and interpreted by the RS. None of the claims defined below are intended to be mandatory to use or implement in all cases, but rather they provide a starting point for a set of useful, interoperable claims. Applications using CWTs should define which specific claims they use and when they are required or optional.

D.1.1. iss (Issuer) Claim

The "iss" (issuer) claim identifies the principal that issued the CWT. The processing of this claim is generally application specific. The "iss" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

D.1.2. sub (Subject) Claim

The "sub" (subject) claim identifies the principal that is the subject of the CWT. The claims in a CWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique. The processing of this claim is generally application specific. The "sub" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

D.1.3. aud (Audience) Claim

The "aud" (audience) claim identifies the recipients that the CWT is intended for. Each principal intended to process the CWT MUST identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the "aud" claim when this claim is present, then the CWT MUST be rejected. In the general case, the "aud" value is an array of case-sensitive strings, each containing a StringOrURI value. In the special case when the CWT has one audience, the "aud" value MAY be a single case-sensitive string containing a StringOrURI value. The interpretation of audience values is generally application specific. Use of this claim is OPTIONAL.

D.1.4. exp (Expiration Time) Claim

The "exp" (expiration time) claim identifies the expiration time on or after which the CWT MUST NOT be accepted for processing. The processing of the "exp" claim requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

D.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim identifies the time before which the CWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

D.1.6. iat (Issued At) Claim

The "iat" (issued at) claim identifies the time at which the CWT was issued. This claim can be used to determine the age of the CWT. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.

D.1.7. cti (CWT ID) Claim

The "cti" (CWT ID) claim provides a unique identifier for the CWT. The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "cti" claim can be used to prevent the CWT from being replayed. The "cti" value is a case-sensitive string. Use of this claim is OPTIONAL.

D.1.8. cnf (Confirmation) Claim

The "cnf" (confirmation) claim is used in the CWT to contain members used to identify a proof-of-possession key. The "cnf" claim is used to express a declaration in a CWT that a Client of the CWT possesses a particular key and that the recipient can cryptographically confirm proof-of-possession of the key by the client.

D.1.9. cks (COSE Key Structure) Claim

The "cks" (COSE Key Structure) claim holds members representing a COSE Key Structure. The members of the structure can be found in Section 7.1 of [I-D.ietf-cose-msg].

D.1.10. aif (Authorization Information Format) Claim

The "aif" (Authorization Information Format) claim uses the AIF format defined in [I-D.bormann-core-ace-aif] to transfer information about the authorization from the AS to the RS.

D.2. CBOR major types for Claims

Value	Major Type	Key
0	0	iss
1	0	sub
2	0	aud
3	0	nonce
4	0	exp

5	0	iat
6	4	cnf
7	0	ck
8	4	aif

Figure 28: CBOR Mappings used in CWT Access Tokens.

Note: Claims defined by the OpenID Foundation have not yet been included in the table above.

D.3. CBOR Web Token Example

This section illustrates a CWT in the CBOR diagnostic notation. This example CWT was issued by the AS identified as "coap://as.example.com" in the "iss" (issuer) claim. The CWT is only valid at a resource server at "coap://light.example.com". It’s validity is 2 minutes and it includes a symmetric key that will be used to secure the communication, either using object security, or transport security, between the client and the resource server. The "aif" claim includes AIF objects that assert that subject is authorized to make a PUT request against the "/s/light" resource, a PUT and a GET against the "/a/led" resource and a POST against the "/dlts" resource.

```
{
  "iss" : "coap://as.example.com",
  "aud" : "coap://light.example.com",
  "exp" : 1444064944,
  "iat" : 1443944944,
  "aif" : [["/s/light", 1], ["/a/led", 5], ["/dtls", 2]],
  "cnf" : {
    "jwk" : b64’JDLUhTMjU2IiwiY3R5Ijoi ...’
  }
}
```

Figure 29: CWT Example in the CBOR Diagnostic Notation.

Authors’ Addresses

Ludwig Seitz
SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
SWEDEN

Email: goran.selander@ericsson.com

Erik Wahlstroem
Nexus Technology
Telefonvagen 26
Hagersten 126 26
Sweden

Email: erik.wahlstrom@nexusgroup.com

Samuel Erdtman
Nexus Technology
Telefonvagen 26
Hagersten 126 26
Sweden

Email: samuel.erdtman@nexusgroup.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com