

ACE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2019

S. Gerdes  
Universitaet Bremen TZI  
L. Seitz  
RISE SICS  
G. Selander  
Ericsson AB  
C. Bormann, Ed.  
Universitaet Bremen TZI  
October 22, 2018

An architecture for authorization in constrained environments  
draft-ietf-ace-actors-07

## Abstract

Constrained-node networks are networks where some nodes have severe constraints on code size, state memory, processing capabilities, user interface, power and communication bandwidth (RFC 7228).

This document provides terminology, and identifies the elements that an architecture needs to address, providing a problem statement, for authentication and authorization in these networks.

## Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

## Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	4
2. Architecture and High-level Problem Statement . . . . .	6
2.1. Elements of an Architecture . . . . .	6
2.2. Architecture Variants . . . . .	9
2.3. Information Flows . . . . .	11
3. Security Objectives . . . . .	13
3.1. End-to-End Security Objectives in Multi-Hop Scenarios . .	13
4. Authentication and Authorization . . . . .	14
5. Actors and their Tasks . . . . .	16
5.1. Constrained Level Actors . . . . .	17
5.2. Principal Level Actors . . . . .	18
5.3. Less-Constrained Level Actors . . . . .	18
6. Kinds of Protocols . . . . .	19
6.1. Constrained Level Protocols . . . . .	20
6.1.1. Cross Level Support Protocols . . . . .	20
6.2. Less-Constrained Level Protocols . . . . .	20
7. Elements of a Solution . . . . .	21
7.1. Authorization . . . . .	21
7.2. Authentication . . . . .	22
7.3. Communication Security . . . . .	22
7.4. Cryptographic Keys . . . . .	23
8. Assumptions and Requirements . . . . .	24
8.1. Constrained Devices . . . . .	24
8.2. Server-side Authorization . . . . .	24
8.3. Client-side Authorization Information . . . . .	25
8.4. Resource Access . . . . .	25
8.5. Keys and Cipher Suites . . . . .	25
8.6. Network Considerations . . . . .	26
9. Security Considerations . . . . .	26
9.1. Physical Attacks on Sensor and Actuator Networks . . . .	27
9.2. Clocks and Time Measurements . . . . .	27
10. IANA Considerations . . . . .	28
11. Informative References . . . . .	28
Acknowledgements . . . . .	30
Authors' Addresses . . . . .	30

## 1. Introduction

As described in [RFC7228], constrained nodes are small devices with limited abilities which in many cases are made to fulfill a specific simple task. They may have limited hardware resources such as processing power, memory, non-volatile storage and transmission capacity and additionally in most cases do not have user interfaces and displays. Due to these constraints, commonly used security protocols are not always easily applicable, or may give rise to particular deployment/management challenges.

As components of the Internet of Things (IoT), constrained nodes are expected to be integrated in all aspects of everyday life and thus will be entrusted with vast amounts of data. Without appropriate security mechanisms attackers might gain control over things relevant to our lives. Authentication and authorization mechanisms are therefore prerequisites for a secure Internet of Things.

Applications generally require some degree of authentication and authorization, which gives rise to some complexity. Authorization is about who can do what to which objects (see also [RFC4949]). Authentication specifically addresses the who, but is often specific to the authorization that is required (for example, it may be sufficient to authenticate the age of an actor, so no identifier is needed or even desired). Authentication often involves credentials, only some of which need to be long-lived and generic; others may be directed towards specific authorizations (but still possibly long-lived). Authorization then makes use of these credentials, as well as other information (such as the time of day). This means that the complexity of authenticated authorization can often be moved back and forth between these two aspects.

In some cases authentication and authorization can be addressed by static configuration provisioned during manufacturing or deployment by means of fixed trust anchors and static access control lists. This is particularly applicable to siloed, fixed-purpose deployments.

However, as the need for flexible access to assets already deployed increases, the legitimate set of authorized entities as well as their specific privileges cannot be conclusively defined during deployment, without any need for change during the lifetime of the device. Moreover, several use cases illustrate the need for fine-grained access control policies, for which for instance a basic access control list concept may not be sufficiently powerful [RFC7744].

The limitations of the constrained nodes impose a need for security mechanisms which take the special characteristics of constrained environments into account; not all constituents may be able to

perform all necessary tasks by themselves. To put it the other way round: the security mechanisms that protect constrained nodes must remain effective and manageable despite the limitations imposed by the constrained environment.

Therefore, in order to be able to achieve complex security objectives between actors some of which are hosted on simple ("constrained") devices, some of the actors will make use of help from other, less constrained actors. (This offloading is not specific to networks with constrained nodes, but their constrainedness as the main motivation is.)

We therefore group the logical functional entities by whether they can be assigned to a constrained device ("constrained level") or need higher function platforms ("less-constrained level"); the latter does not necessarily mean high-function, "server" or "cloud" platforms. Note that assigning a logical functional entity to the constrained level does not mean that the specific implementation needs to be constrained, only that it can be.

The description assumes that some form of setup (aspects of which are often called provisioning and/or commissioning) has already been performed and at least some initial security relationships important for making the system operational have already been established.

This document provides some terminology, and identifies the elements an architecture needs to address, representing the relationships between the logical functional entities involved; on this basis, a problem description for authentication and authorization in constrained-node networks is provided.

### 1.1. Terminology

Readers are assumed to be familiar with the terms and concepts defined in [RFC4949], including "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

REST terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252]; the latter also defines additional terms such as "endpoint".

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. is defined in [RFC7228].

In addition, this document uses the following terminology:

Resource (R): an item of interest which is represented through an interface. It might contain sensor or actuator values or other information. (Intended to coincide with the definitions of [RFC7252] and [RFC7231].)

Constrained node: a constrained device in the sense of [RFC7228].

Actor: A logical functional entity that performs one or more tasks. Multiple actors may be present within a single device or a single piece of software.

Resource Server (RS): An entity which hosts and represents a Resource. (Used here to discuss the server that provides a resource that is the end, not the means, of the authenticated authorization process - i.e., not CAS or AS.)

Client (C): An entity which attempts to access a resource on a RS. (Used to discuss the client whose access to a resource is the end, not the means, of the authenticated authorization process.)

Overseeing principal: (Used in its English sense here, and specifically as:) An individual that is either RqP or RO or both.

Resource Owner (RO): The overseeing principal that is in charge of the resource and controls its access permissions.

Requesting Party (RqP): The overseeing principal that is in charge of the Client and controls the requests a Client makes and its acceptance of responses.

Authorization Server (AS): An entity that prepares and endorses authentication and authorization data for a Resource Server.

Client Authorization Server (CAS): An entity that prepares and endorses authentication and authorization data for a Client.

Authorization Manager: An entity that prepares and endorses authentication and authorization data for a constrained node. Used in constructions such as "a constrained node's authorization manager" to denote AS for RS and CAS for C.

Authenticated Authorization: The confluence of mechanisms for authentication and authorization, ensuring that authorization is applied to and made available for authenticated entities and that entities providing authentication services are authorized to do so for the specific authorization process at hand.

Note that other authorization architectures such as OAuth [RFC6749] or UMA [I-D.hardjono-oauth-umacore] focus on the authorization problems on the RS side, in particular what accesses to resources the RS is to allow. In this document the term authorization includes this aspect, but is also used for the client-side aspect of authorization, i.e., more generally allowing RqPs to decide what interactions clients may perform with other endpoints.

## 2. Architecture and High-level Problem Statement

This document deals with how to control and protect resource-based interaction between potentially constrained endpoints. The following setting is assumed as a high-level problem statement:

- o An endpoint may host functionality of one or more actors.
- o C in one endpoint requests to access R on a RS in another endpoint.
- o A priori, the endpoints do not necessarily have a pre-existing security relationship to each other.
- o Either of the endpoints, or both, may be constrained.

### 2.1. Elements of an Architecture

In its simplest expression, the architecture starts with a two-layer model: the principal level (at which components are assumed to be functionally unconstrained) and the constrained level (at which some functional constraints are assumed to apply to the components).

Without loss of generality, we focus on the C functionality in one endpoint, which we therefore also call C, accessing the RS functionality in another endpoint, which we therefore also call RS.

The constrained level and its security objectives are detailed in Section 5.1.

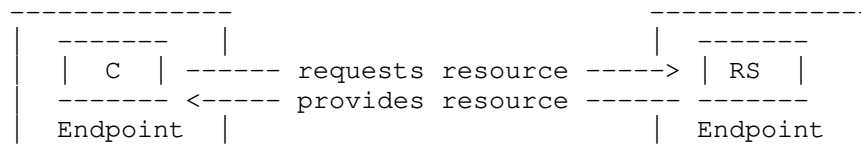


Figure 1: Constrained Level

The authorization decisions at the endpoints are made on behalf of the overseeing principals that control the endpoints. To reuse OAuth and UMA terminology, the present document calls the overseeing principal that is controlling C the Requesting Party (RqP), and calls the overseeing principal that is controlling RS the Resource Owner (RO). Each overseeing principal makes authorization decisions (possibly encapsulating them into security policies) which are then enforced by the endpoint it controls.

The specific security objectives will vary, but for any specific version of this scenario will include one or more of:

- o Objectives of type 1: No entity not authorized by the RO has access to (or otherwise gains knowledge of) R.
- o Objectives of type 2: C is exchanging information with (sending a request to, accepting a response from) a resource only where it can ascertain that RqP has authorized the exchange with R.

Objectives of type 1 require performing authorization on the Resource Server side while objectives of type 2 require performing authorization on the Client side.

More on the security objectives of the principal level in Section 5.2.

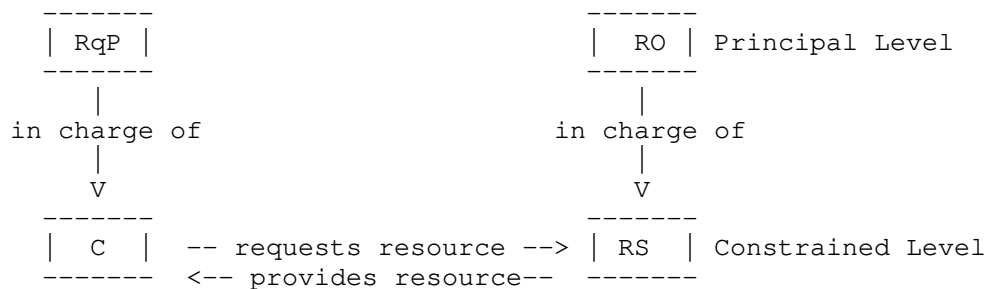


Figure 2: Constrained Level and Principal Level

The use cases defined in [RFC7744] demonstrate that constrained devices are often used for scenarios where their overseeing principals are not present at the time of the communication, are not able to communicate directly with the device because of a lack of user interfaces or displays, or may prefer the device to communicate autonomously.

Moreover, constrained endpoints may need support with tasks requiring heavy processing, large memory or storage, or interfacing to humans,

such as management of security policies defined by an overseeing principal. The principal, in turn, requires some agent maintaining the policies governing how its endpoints will interact.

For these reasons, another level of nodes is introduced in the architecture, the less-constrained level (illustrated below in Figure 3). Using OAuth terminology, AS acts on behalf of the RO to control and support the RS in handling access requests, employing a pre-existing security relationship with RS. We complement this with CAS acting on behalf of RqP to control and support the C in making resource requests and acting on the responses received, employing a pre-existing security relationship with C. To further relieve the constrained level, authorization (and related authentication) mechanisms may be employed between CAS and AS (Section 6.2). (Again, both CAS and AS are conceptual entities controlled by their respective overseeing principals. Many of these entities, often acting for different overseeing principals, can be combined into a single server implementation; this of course requires proper segregation of the control information provided by each overseeing principal.)

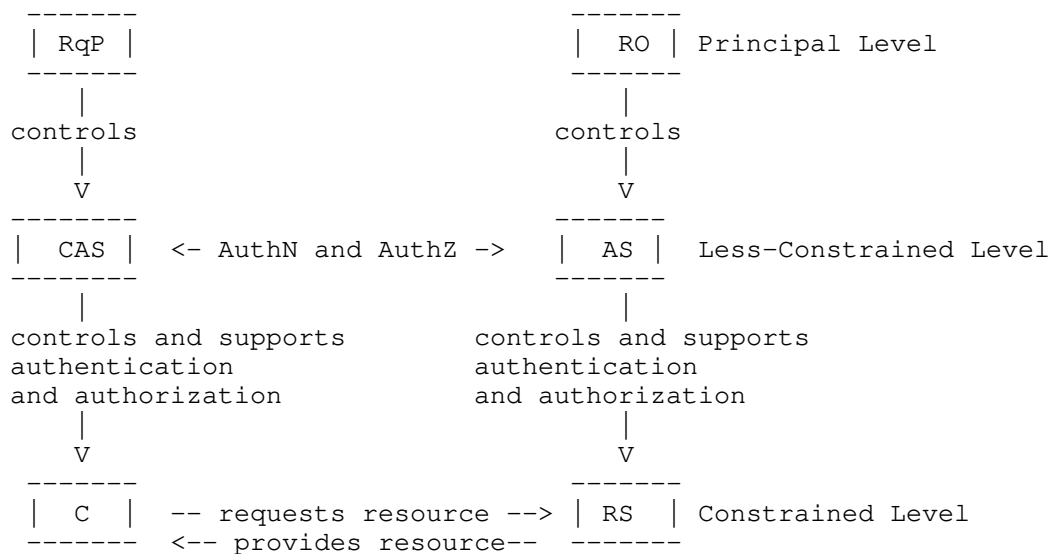


Figure 3: Overall architecture

Figure 3 shows all three levels considered in this document. Note that the vertical arrows point down to illustrate exerting control and providing support; this is complemented by information flows that often are bidirectional. Note also that not all entities need to be ready to communicate at any point in time; for instance, RqP may have



provided enough information to CAS that CAS can autonomously negotiate access to RS with AS for C based on this information.

## 2.2. Architecture Variants

The elements of the architecture described above are indeed architectural; that is, they are parts of a conceptual model, and may be instantiated in various ways in practice. For example, in a given scenario, several elements might share a single device or even be combined in a single piece of software. If C is located on a more powerful device, it can be combined with CAS:

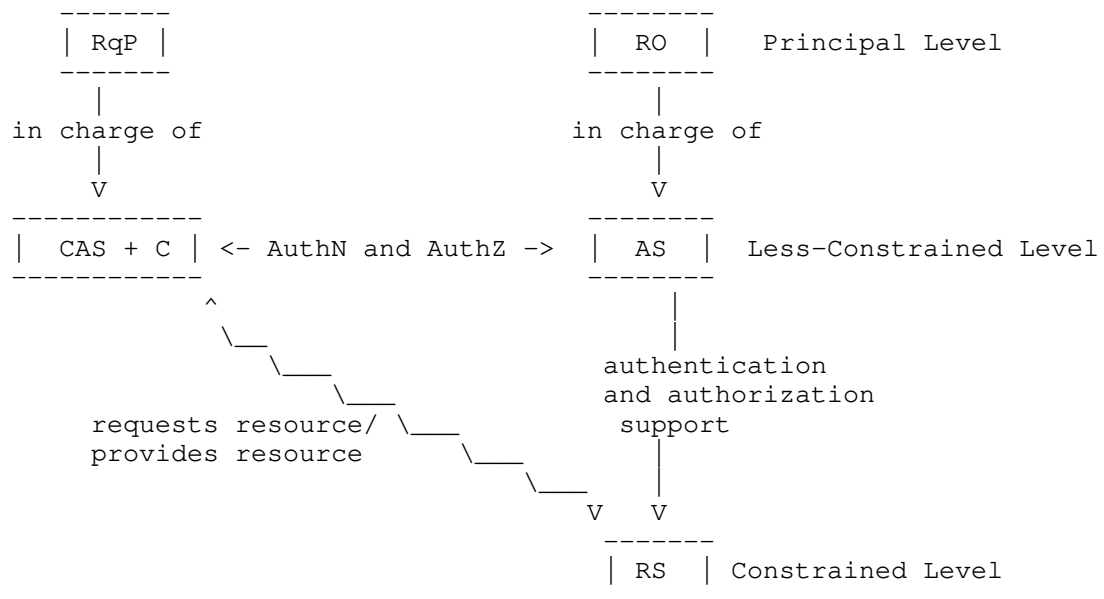


Figure 4: Combined C and CAS

If RS is located on a more powerful device, it can be combined with AS:

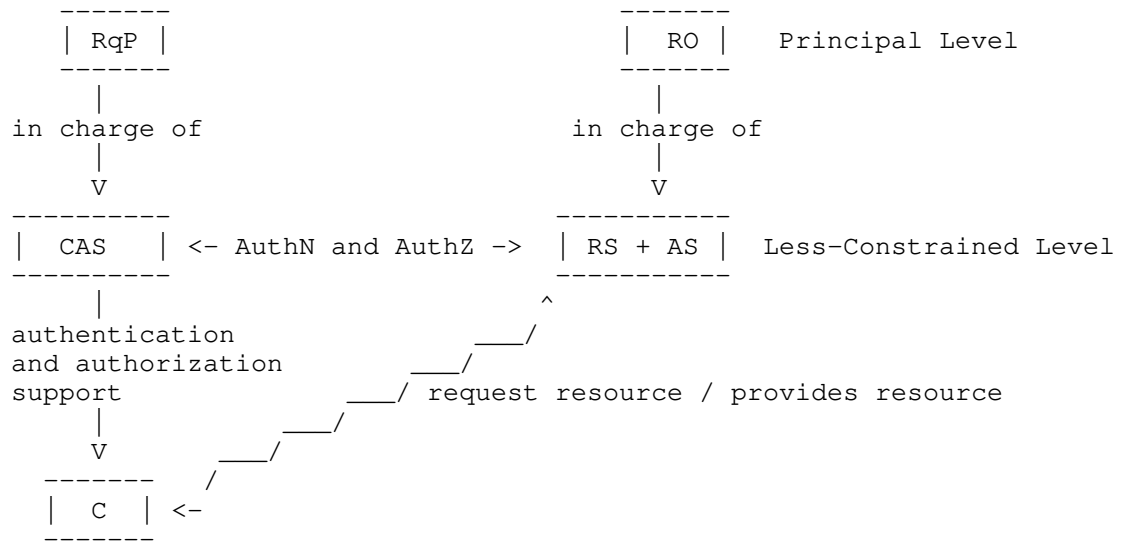


Figure 5: Combined AS and RS

If C and RS have the same overseeing principal, CAS and AS can be combined.

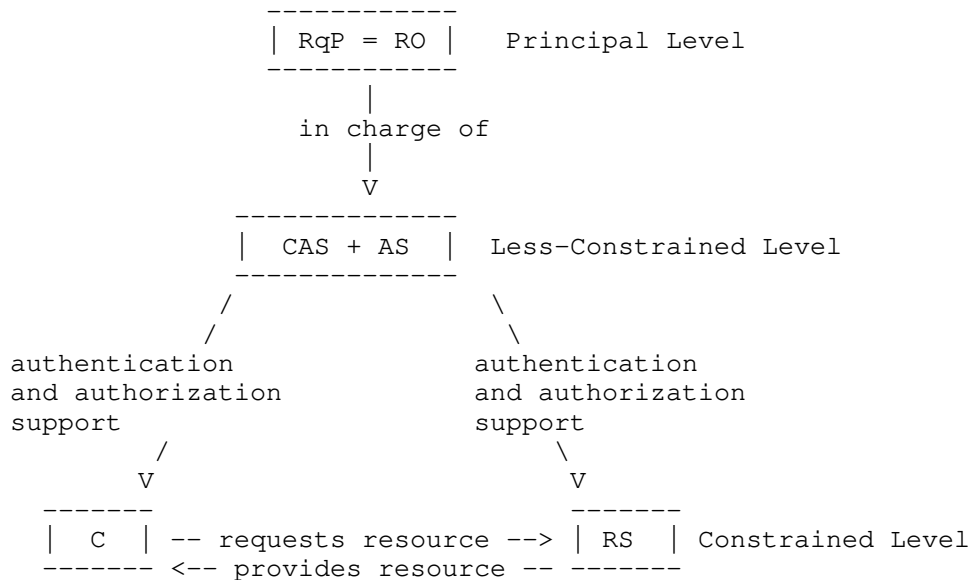


Figure 6: CAS combined with AS

### 2.3. Information Flows

We now formulate the problem statement in terms of the information flows the architecture focuses on. (While the previous section discusses the architecture in terms of abstract devices and their varying roles, the actual protocols being standardized define those information flows and the messages embodying them: "RESTful architectures focus on defining interfaces and not components" ([REST], p. 116).)

The interaction with the nodes on the principal level, RO and RqP, is not involving constrained nodes and therefore can employ an existing mechanism. The less-constrained nodes, CAS and AS, support the constrained nodes, C and RS, with control information, for example permissions of clients, conditions on resources, attributes of client and resource servers, keys and credentials. This control information may be rather different for C and RS.

The potential information flows are shown in Figure 7. The direction of the vertical arrows expresses the exertion of control; actual information flow is bidirectional.

The message flow may pass unprotected paths and thus need to be protected, potentially beyond a single REST hop (Section 3.1):

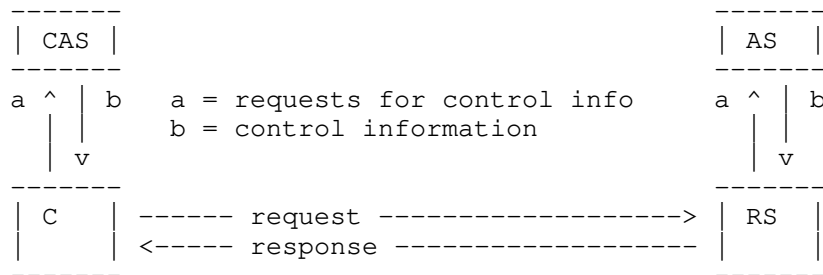


Figure 7: Information flows that need to be protected

- o We assume that the necessary keys/credentials for protecting the control information between the potentially constrained nodes and their associated less-constrained nodes are pre-established, for example as part of the commissioning procedure.
- o Any necessary keys/credentials for protecting the interaction between the potentially constrained nodes will need to be established and maintained as part of a solution.

In terms of the elements of the architecture laid out above, this document's problem statement for authorization in constrained environments can then be summarized as follows:

- o The interaction between potentially constrained endpoints is controlled by control information provided by less-constrained nodes on behalf of the overseeing principals of the endpoints.
- o The interaction between the endpoints needs to be secured, as well as the establishment of the necessary keys for securing the interaction, potentially end-to-end through intermediary nodes.
- o The mechanism for transferring control information needs to be secured, potentially end-to-end through intermediary nodes. Pre-established keying material may need to be employed for establishing the keys used to protect these information flows.

(Note that other aspects relevant to secure constrained node communication such as secure bootstrap or group communication are not specifically addressed by the present document.)

### 3. Security Objectives

The security objectives that are addressed by an authorization solution include confidentiality and integrity. Additionally, an authorization solution has an impact on the availability: First, by reducing the load (only accepting selected operations by selected entities limits the burden on system resources), and second, because misconfigured or wrongly designed authorization solutions can result in availability breaches (denial of service) as users might no longer be able to use data and services as they are supposed to.

Authentication mechanisms can help achieve additional security objectives such as accountability and third-party verifiability. These additional objectives are not directly related to authorization and thus are not in scope of this draft, but may nevertheless be relevant. Accountability and third-party verifiability may require authentication on a device level, if it is necessary to determine which device performed an action. In other cases it may be more important to find out who is responsible for the device's actions. (The ensuing requirements for logging, auditability, and the related integrity requirements are very relevant for constrained devices as well, but outside the scope of this document.) See also Section 4 for more discussion about authentication and authorization.

The security objectives and their relative importance differ for the various constrained environment applications and use cases [RFC7744].

The architecture is based on the observation that different parties may have different security objectives. There may also be a "collaborative" dimension: to achieve a security objective of one party, another party may be required to provide a service. For example, if RqP requires the integrity of representations of a resource R that RS is hosting, both C and RS need to partake in integrity-protecting the transmitted data. Moreover, RS needs to protect any write access to this resource as well as to relevant other resources (such as configuration information, firmware update resources) to prevent unauthorized users from manipulating R.

#### 3.1. End-to-End Security Objectives in Multi-Hop Scenarios

In many cases, the information flows described in Section 2.3 cross multiple client-server pairings but still need to be protected end-to-end. For example, AS may not be connected to RS (or may not want to exercise such a connection), relying on C for transferring authorization information. As the authorization information is related to the permissions granted to C, C must not be in a position to manipulate this information, which therefore requires integrity protection on the way between AS and RS.

As another example, resource representations sent between endpoints may be stored in intermediary nodes, such as caching proxies or pub-sub brokers. Where these intermediaries cannot be relied on to fulfill the security objectives of the endpoints, it is the endpoints that will need to protect the exchanges beyond a single client-server exchange.

Note that there may also be cases of intermediary nodes that very much partake in the security objectives to be achieved. The question what are the pairs of endpoints between which the communication needs end-to-end protection (and which aspect of protection) is defined by the specific use case. Two examples of intermediary nodes executing security functionality:

- o To enable a trustworthy publication service, a pub-sub broker may be untrusted with the plaintext content of a publication (confidentiality), but required to verify that the publication is performed by claimed publisher and is not a replay of an old publication (authenticity/integrity).
- o To comply with requirements of transparency, a gateway may be allowed to read, verify (authenticity) but not modify (integrity) a resource representation which therefore also is end-to-end integrity protected from the server towards a client behind the gateway.

In order to support the required communication and application security, keying material needs to be established between the relevant nodes in the architecture.

#### 4. Authentication and Authorization

Server-side authorization solutions aim at protecting the access to items of interest, for instance hardware or software resources or data: They enable the resource owner to control who can access it and how.

To determine if an entity is authorized to access a resource, an authentication mechanism is needed. According to the Internet Security Glossary [RFC4949], authentication is "the process of verifying a claim that a system entity or system resource has a certain attribute value." Examples for attribute values are the ID of a device, the type of the device or the name of its owner.

The security objectives the authorization mechanism aims at can only be achieved if the authentication and the authorization mechanism work together correctly. We speak of authenticated authorization to

refer to the required synthesis of mechanisms for authentication and authorization.

Where used for authorization, the set of authenticated attributes must be meaningful for this purpose, i.e., authorization decisions must be possible based on these attributes. If the authorization policy assigns permissions to an individual entity, the set of authenticated attributes must be suitable to uniquely identify this entity.

In scenarios where devices are communicating autonomously there is often less need to uniquely identify an individual device: For an overseeing principal, the fact that a device belongs to a certain company or that it has a specific type (such as a light bulb) or location may be more important than that it has a unique identifier.

Overseeing principals (RqP and RO) need to decide about the required level of granularity for the authorization. For example, we distinguish device authorization from owner authorization, and binary authorization from unrestricted authorization. In the first case different access permissions are granted to individual devices while in the second case individual owners are authorized. If binary authorization is used, all authenticated entities are implicitly authorized and have the same access permissions. Unrestricted authorization for an item of interest means that no authorization mechanism is used for accessing this resource (not even by authentication) and all entities are able to access the item as they see fit (note that an authorization mechanism may still be used to arrive at the decision to employ unrestricted authorization).

Authorization granularity	Authorization is contingent on:
device	authentication of specific device
owner	(authenticated) authorization by owner
binary	(any) authentication
unrestricted	(unrestricted access; access always authorized)

Table 1: Some granularity levels for authorization

More fine-grained authorization does not necessarily provide more security but can be more flexible. Overseeing principals need to consider that an entity should only be granted the permissions it

really needs (principle of least privilege), to ensure the confidentiality and integrity of resources.

Client-side authorization solutions aim at protecting the client from disclosing information to or ingesting information from resource servers RqP does not want it to interact with in the given way. Again, binary authorization (the server can be authenticated) may be sufficient, or more fine-grained authorization may be required. The client-side authorization also pertains to the level of protection required for the exchanges with the server (e.g., confidentiality). In the browser web, client-side authorization is often left to the human user that directly controls the client; a constrained client may not have that available all the time but still needs to implement the wishes of the overseeing principal controlling it, the RqP.

For the cases where an authorization solution is needed (all but unrestricted authorization), the enforcing party needs to be able to authenticate the party that is to be authorized. Authentication is therefore required for messages that contain (or otherwise update) representations of an accessed item. More precisely: The enforcing party needs to make sure that the receiver of a message containing a representation is authorized to receive it, both in the case of a client sending a representation to a server and vice versa. In addition, it needs to ensure that the actual sender of a message containing a representation is indeed the one authorized to send this message, again for both the client-to-server and server-to-client case. To achieve this, integrity protection of these messages is required: Authenticity of the message cannot be assured if it is possible for an attacker to modify it during transmission.

In some cases, only one side (client or server side) requires the integrity and / or confidentiality of a resource value. Overseeing principals may decide to omit authentication (unrestricted authorization), or use binary authorization (just employing an authentication mechanism). However, as indicated in Section 3, the security objectives of both sides must be considered, which can often only be achieved when the other side can be relied on to perform some security service.

## 5. Actors and their Tasks

This and the following section look at the resulting architecture from two different perspectives: This section provides a more detailed description of the various "actors" in the architecture, the logical functional entities performing the tasks required. The following section then will focus on the protocols run between these functional entities.



For the purposes of this document, an actor consists of a set of tasks and additionally has a security domain (client domain or server domain) and a level (constrained, principal, less-constrained). Tasks are assigned to actors according to their security domain and required level.

Note that actors are a concept to understand the security requirements for constrained devices. The architecture of an actual solution might differ as long as the security requirements that derive from the relationship between the identified actors are considered. Several actors might share a single device or even be combined in a single piece of software. Interfaces between actors may be realized as protocols or be internal to such a piece of software.

#### 5.1. Constrained Level Actors

As described in the problem statement (see Section 2), either C or RS or both of them may be located on a constrained node. We therefore define that C and RS must be able to perform their tasks even if they are located on a constrained node. Thus, C and RS are considered to be Constrained Level Actors.

C performs the following tasks:

- o Communicate in a secure way (provide for confidentiality and integrity of messages), including access requests.
- o Validate that the RqP ("client-side") authorization information allows C to communicate with RS as a server for R (i.e., from C's point of view, RS is authorized as a server for the specific access to R).

RS performs the following tasks:

- o Communicate in a secure way (provide for confidentiality and integrity of messages), including responses to access requests.
- o Validate that the RO ("server-side") authorization information allows RS to grant C access to the requested resource as requested (i.e., from RS' point of view, C is authorized as a client for the specific access to R).

R is an item of interest such as a sensor or actuator value. R is considered to be part of RS and not a separate actor. The device on which RS is located might contain several resources controlled by different ROs. For simplicity of exposition, these resources are described as if they had separate RS.

As C and RS do not necessarily know each other they might belong to different security domains.

(See Figure 8.)

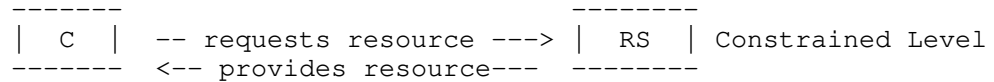


Figure 8: Constrained Level Actors

## 5.2. Principal Level Actors

Our objective is that C and RS are under control of overseeing principals in the physical world, the Requesting Party (RqP) and the Resource Owner (RO) respectively. The overseeing principals decide about the security policies of their respective endpoints; each overseeing principal belongs to the same security domain as their endpoints.

RqP is in charge of C, i.e. RqP specifies security policies for C, such as with whom C is allowed to communicate. By definition, C and RqP belong to the same security domain.

RqP must fulfill the following task:

- o Configure for C authorization information for sources for R.

RO is in charge of R and RS. RO specifies authorization policies for R and decides with whom RS is allowed to communicate. By definition, R, RS and RO belong to the same security domain.

RO must fulfill the following task:

- o Configure for RS authorization information for accessing R.

(See Figure 2.)

## 5.3. Less-Constrained Level Actors

Constrained level actors can only fulfill a limited number of tasks and may not have network connectivity all the time. To relieve them from having to manage keys for numerous endpoints and conducting computationally intensive tasks, another level of complexity for actors is introduced (and, thus, a stricter limit on their constrainedness). An actor on the less-constrained level belongs to

the same security domain as its respective constrained level actor. They also have the same overseeing principal.

The Client Authorization Server (CAS) belongs to the same security domain as C and RqP. CAS acts on behalf of RqP. It assists C in authenticating RS and determining if RS is an authorized server for R. CAS can do that because for C, CAS is the authority for claims about RS.

CAS performs the following tasks:

- o Vouch for the attributes of its clients.
- o Ascertain that C's overseeing principal (RqP) authorized AS to vouch for RS and provide keying material for it.
- o Provide revocation information concerning its clients (optional).
- o Obtain authorization information about RS from C's overseeing principal (RqP) and provide it to C.
- o Negotiate means for secure communication to communicate with C.

The Authorization Server (AS) belongs to the same security domain as R, RS and RO. AS acts on behalf of RO. It supports RS by authenticating C and determining C's permissions on R. AS can do that because for RS, AS is the authority for claims about C.

AS performs the following tasks:

- o Vouch for the attributes of its resource servers.
- o Ascertain that RS's overseeing principal (RO) authorized CAS to vouch for C and provide keying material for it.
- o Provide revocation information concerning its servers (optional).
- o Obtain authorization information about C from RS' overseeing principal (RO) and provide it to RS.
- o Negotiate means for secure communication to communicate with RS.

## 6. Kinds of Protocols

Devices on the less-constrained level potentially are more powerful than constrained level devices in terms of processing power, memory, non-volatile storage. This results in different characteristics for the protocols used on these levels.

### 6.1. Constrained Level Protocols

A protocol is considered to be on the constrained level if it is used between the actors C and RS which are considered to be constrained (see Section 5.1). C and RS might not belong to the same security domain. Therefore, constrained level protocols need to work between different security domains.

Commonly used Internet protocols can not in every case be applied to constrained environments. In some cases, tweaking and profiling is required. In other cases it is beneficial to define new protocols which were designed with the special characteristics of constrained environments in mind.

On the constrained level, protocols need to address the specific requirements of constrained environments. Examples for protocols that consider these requirements is the transfer protocol CoAP (Constrained Application Protocol) [RFC7252] and the Datagram Transport Layer Security Protocol (DTLS) [RFC6347] which can be used for channel security.

Constrained devices have only limited storage space and thus cannot store large numbers of keys. This is especially important because constrained networks are expected to consist of thousands of nodes. Protocols on the constrained level should keep this limitation in mind.

#### 6.1.1. Cross Level Support Protocols

We refer to protocols that operate between a constrained device and its corresponding less-constrained device as cross-level support protocols. Protocols used between C and CAS or RS and AS are therefore support protocols.

Support protocols must consider the limitations of their constrained endpoint and therefore belong to the constrained level protocols.

### 6.2. Less-Constrained Level Protocols

A protocol is considered to be on the less-constrained level if it is used between the actors CAS and AS. CAS and AS might belong to different security domains.

On the less-constrained level, HTTP [RFC7230] and Transport Layer Security (TLS) [RFC8246] can be used alongside or instead of CoAP and DTLS. Moreover, existing security solutions for authentication and authorization such as the OAuth web authorization framework [RFC6749] and Kerberos [RFC4120] can likely be used without modifications and

the less-constrained layer is assumed to impose no constraints that would inhibit the traditional deployment/use of, e.g., a Public Key Infrastructure (PKI).

## 7. Elements of a Solution

Without anticipating specific solutions, the following considerations may be helpful in discussing them.

### 7.1. Authorization

The core problem we are trying to solve is authorization. The following problems related to authorization need to be addressed:

- o AS needs to transfer authorization information to RS and CAS needs to transfer authorization information to C.
- o The transferred authorization information needs to follow a defined format and encoding, which must be efficient for constrained devices, considering size of authorization information and parser complexity.
- o C and RS need to be able to verify the authenticity of the authorization information they receive. C must ascertain that the authorization information stems from a CAS that was authorized by RqP, RS must validate that the authorization information stems from an AS that was authorized by RO.
- o Some applications may require the confidentiality of authorization information. It then needs to be encrypted between CAS and C and AS and RS, respectively.
- o C and RS must be able to check the freshness of the authorization information and determine for how long it is supposed to be valid.
- o The RS needs to enforce the authorization decisions of the AS, while C needs to abide with the authorization decisions of the CAS. The authorization information might require additional policy evaluation (such as matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the constrained actors needs to be adapted to the capabilities of the devices implementing them.
- o Finally, as is indicated in the previous bullet, for a particular authorization decision there may be different kinds of authorization information needed, and these pieces of information may be transferred to C and RS at different times and in different ways prior to or during the client request.

## 7.2. Authentication

The following problems need to be addressed, when considering authentication:

- o RS needs to authenticate AS in the sense that it must be certain that it communicates with an AS that was authorized by RO, C needs to authenticate CAS in the sense that it must be certain that it communicates with a CAS that was authorized by RqP, to ensure that the authorization information and related data comes from the correct source.
- o C must securely have obtained keying material to communicate with its CAS that is up to date and that is updated if necessary. RS must securely have obtained keying material to communicate with AS that is up to date and that is updated if necessary.
- o CAS and AS may need to authenticate each other, both to perform the required business logic and to ensure that CAS gets security information related to the resources from the right source.
- o In some use cases RS needs to authenticate some property of C, in order to map it to the relevant authorization information.
- o C may need to authenticate RS, in order to ensure that it is interacting with the right resources.
- o CAS and AS need to authenticate their communication partner (C or RS), in order to ensure it serves the correct device. If C and AS vouch for keying material or certain attributes of their respective constrained devices, they must ascertain that the devices actually currently have this keying material or these attributes.

## 7.3. Communication Security

There are different alternatives to provide communication security, and the problem here is to choose the optimal one for each scenario. We list the available alternatives:

- o Session-based security at transport layer such as DTLS [RFC6347] offers security, including integrity and confidentiality protection, for the whole application layer exchange. However, DTLS may not provide end-to-end security over multiple hops. Another problem with DTLS is the cost of the handshake protocol, which may be too expensive for constrained devices especially in terms of memory and power consumption for message transmissions.

- o An alternative is object security at application layer, for instance using [I-D.ietf-core-object-security]. Secure objects can be stored or cached in network nodes and provide security for a more flexible communication model such as publish/subscribe (compare e.g. CoRE Mirror Server [I-D.ietf-core-coap-pubsub]). A problem with object security is that it can not provide confidentiality for the message headers.
- o Hybrid solutions using both session-based and object security are also possible. An example of a hybrid is where authorization information and cryptographic keys are provided by AS in the format of secure data objects, but where the resource access is protected by session-based security.

#### 7.4. Cryptographic Keys

With respect to cryptographic keys, we see the following problems that need to be addressed:

##### Symmetric vs Asymmetric Keys

We need keys both for protection of resource access and for protection of transport of authentication and authorization information. It may be necessary to support solutions that require the use of asymmetric keys as well as ones that get by with symmetric keys, in both cases. There are classes of devices that can easily perform symmetric cryptography, but consume considerably more time/battery for asymmetric operations. On the other hand asymmetric cryptography has benefits such as in terms of deployment.

##### Key Establishment

How are the corresponding cryptographic keys established? Considering Section 7.1 there must be a mapping between these keys and the authorization information, at least in the sense that AS must be able to specify a unique client identifier which RS can verify (using an associated key). One of the use cases of [RFC7744] describes spontaneous change of access policies - such as giving a hitherto unknown client the right to temporarily unlock your house door. In this case C is not previously known to RS and a key must be provisioned by AS.

##### Revocation and Expiration

How are keys replaced and how is a key that has been compromised revoked in a manner that reaches all affected parties, also keeping in mind scenarios with intermittent connectivity?

## 8. Assumptions and Requirements

In this section we list a set of candidate assumptions and requirements to make the problem description in the previous sections more concise and precise. Note that many of these assumptions and requirements are targeting specific solutions and not the architecture itself.

### 8.1. Constrained Devices

- o C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover:
  - \* unable to manage complex authorization policies
  - \* unable to manage a large number of secure connections
  - \* without user interface
  - \* without constant network connectivity
  - \* unable to precisely measure time
  - \* required to save on wireless communication due to high power consumption
- o CAS and AS are not assumed to be constrained devices.
- o All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.
- o Public key cryptography requires additional resources (such as RAM, ROM, power, specialized hardware).
- o A solution will need to consider support for a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. Section 9.2).

### 8.2. Server-side Authorization

- o RS enforces authorization for access to a resource based on credentials presented by C, the requested resource, the REST method, and local context in RS at the time of the request, or on any subset of this information.
- o The authorization decision is enforced by RS.



- \* RS needs to have authorization information in order to verify that C is allowed to access the resource as requested.
- \* RS needs to make sure that it provides resource access only to authorized clients.
- o Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (for instance, resource descriptions).

### 8.3. Client-side Authorization Information

- o C enforces client-side authorization by protecting its requests to RS and by authenticating results from RS, making use of decisions and policies as well as keying material provided by CAS.

### 8.4. Resource Access

- o Resources are accessed in a RESTful manner using methods such as GET, PUT, POST, DELETE.
- o By default, the resource request needs to be integrity protected and may be encrypted end-to-end from C to RS. It needs to be possible for RS to detect a replayed request.
- o By default, the response to a request needs to be integrity protected and may be encrypted end-to-end from RS to C. It needs to be possible for C to detect a replayed response.
- o RS needs to be able to verify that the request comes from an authorized client.
- o C needs to be able to verify that the response to a request comes from the intended RS.
- o There may be resources whose access need not be protected (e.g. for discovery of the responsible AS).

### 8.5. Keys and Cipher Suites

- o A constrained node and its authorization manager (i.e., RS and AS, and C and CAS) have established cryptographic keys. For example, they share a secret key or each have the other's public key.
- o The transfer of authorization information is protected with symmetric and/or asymmetric keys.

- o The access request/response is protected with symmetric and/or asymmetric keys.
- o There must be a mechanism for RS to establish the necessary key(s) to verify and decrypt the request and to protect the response.
- o There must be a mechanism for C to establish the necessary key(s) to protect the request and to verify and decrypt the response.
- o There must be a mechanism for C to obtain the supported cipher suites of a RS.

#### 8.6. Network Considerations

- o A solution will need to consider network overload due to avoidable communication of a constrained node with its authorization manager (C with CAS, RS with AS).
- o A solution will need to consider network overload by compact authorization information representation.
- o A solution may want to optimize the case where authorization information does not change often.
- o A solution should combine the mechanisms for providing authentication and authorization information to the client and RS where possible.
- o A solution may consider support for an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

#### 9. Security Considerations

This document discusses authorization-related tasks for constrained environments and describes how these tasks can be mapped to actors in the architecture.

In this section we focus on specific security aspects related to authorization in constrained-node networks. Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications of specific constraints on the security mechanisms employed. A wider view of security in constrained-node networks is provided in [I-D.irtf-t2trg-iot-secons].

### 9.1. Physical Attacks on Sensor and Actuator Networks

The focus of this work is on constrained-node networks consisting of connected constrained devices such as sensors and actuators. The main function of such devices is to interact with the physical world by gathering information or performing an action. We now discuss attacks performed with physical access to such devices.

The main threats to sensors and actuator networks are:

- o Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.
- o Denial-of-service making the sensor/actuator unable to perform its intended task correctly.

A number of attacks can be made with physical access to a device including probing attacks, timing attacks, power attacks, etc. However, with physical access to a sensor or actuator device it is possible to directly perform attacks equivalent of eavesdropping, manipulating data or denial of service. These attacks are possible by having physical access to the device, since the assets are related to the physical world. Moreover, this kind of attacks are in many cases straightforward (requires no special competence or tools, low cost given physical access, etc). If an attacker has full physical access to a sensor or actuator device, then much of the security functionality elaborated in this draft may not be effective to protect the asset during the physical attack.

### 9.2. Clocks and Time Measurements

Measuring time and keeping wall-clock time with certain accuracy is important to achieve certain security properties, for example to determine whether keying material an access token, or some other assertion, is valid. The required level of accuracy may differ for different applications.

Dynamic authorization in itself requires the ability to handle expiry or revocation of authorization decisions or to distinguish new authorization decisions from old.

For certain categories of devices we can assume that there is an internal clock which is sufficiently accurate to handle the time measurement requirements. If RS continuously measures time and can connect directly to AS, this relationship can be used to update RS in terms of time, removing some uncertainty, as well as to directly provide revocation information, removing authorizations that are no longer desired.

If RS continuously measures time but can't connect to AS or another trusted source of time, time drift may have to be accepted and it may be harder to manage revocation. However, RS may still be able to handle short lived access rights within some margins, by measuring the time since arrival of authorization information or request.

Some categories of devices in scope may be unable to measure time with any accuracy (e.g. because of sleep cycles). This category of devices is not suitable for the use cases which require measuring validity of assertions and authorizations in terms of absolute time such as TLS certificates but require a mechanism that is specifically designed for them.

#### 10. IANA Considerations

This document has no actions for IANA.

#### 11. Informative References

[HUM14delegation]

Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", 11th IEEE International Conference on Sensing, Communication, and Networking (SECON'14), June 30 - July 3, 2014.

[I-D.hardjono-oauth-umacore]

Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-14 (work in progress), January 2016.

[I-D.ietf-core-coap-pubsub]

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

[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.irtf-t2trg-iot-secons]  
Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-secons-15 (work in progress), May 2018.
- [REST] Fielding, R. and R. Taylor, "Principled design of the modern Web architecture", ACM Trans. Inter. Tech. Vol. 2(2), pp. 115-150, DOI 10.1145/514183.514185, May 2002.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [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>>.
- [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>>.

- [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>>.
- [RFC8246] McManus, P., "HTTP Immutable Responses", RFC 8246, DOI 10.17487/RFC8246, September 2017, <<https://www.rfc-editor.org/info/rfc8246>>.

#### Acknowledgements

The authors would like to thank Olaf Bergmann, Robert Cragie, Samuel Erdtman, Klaus Hartke, Sandeep Kumar, John Mattson, Corinna Schmitt, Mohit Sethi, Abhinav Somaraju, Hannes Tschofenig, Vlasios Tsiatsis and Erik Wahlstroem for contributing to the discussion, giving helpful input and commenting on previous forms of this draft. The authors would also like to specifically acknowledge input provided by Hummen and others [HUM14delegation]. Robin Wilton provided extensive editorial comments that were the basis for significant improvements of the text.

#### Authors' Addresses

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

Phone: +49-421-218-63906  
Email: [gerdes@tzi.org](mailto:gerdes@tzi.org)

Ludwig Seitz  
RISE SICS  
Scheelevaegen 17  
Lund 223 70  
Sweden

Email: [ludwig.seitz@ri.se](mailto:ludwig.seitz@ri.se)

Goeran Selander  
Ericsson AB

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

Carsten Bormann (editor)  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

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

ACE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 28, 2016

L. Seitz  
SICS  
G. Selander  
Ericsson  
E. Wahlstroem  
S. Erdtman  
Nexus Technology  
H. Tschofenig  
ARM Ltd.  
February 25, 2016

Authorization for the Internet of Things using OAuth 2.0  
draft-ietf-ace-oauth-authz-01

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

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

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

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

This Internet-Draft will expire on August 28, 2016.

Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of



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

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Overview . . . . .	4
3.1. OAuth 2.0 . . . . .	5
3.2. CoAP . . . . .	7
3.3. Object Security . . . . .	8
4. Protocol Interactions . . . . .	9
5. OAuth 2.0 Profiling . . . . .	12
5.1. Client Information . . . . .	12
5.2. CoAP Access-Token Option . . . . .	15
5.3. Authorization Information Resource at the Resource Server	15
5.3.1. Authorization Information Request . . . . .	16
5.3.2. Authorization Information Response . . . . .	16
5.3.2.1. Success Response . . . . .	16
5.3.2.2. Error Response . . . . .	16
5.4. Authorization Information Format . . . . .	17
5.5. CBOR Data Formats . . . . .	17
5.6. Token Expiration . . . . .	17
6. Deployment Scenarios . . . . .	18
6.1. Client and Resource Server are Offline . . . . .	19
6.2. Resource Server Offline . . . . .	22
6.3. Token Introspection with an Offline Client . . . . .	26
6.4. Always-On Connectivity . . . . .	30
6.5. Token-less Authorization . . . . .	31
6.6. Securing Group Communication . . . . .	34
7. Security Considerations . . . . .	35
8. IANA Considerations . . . . .	35
8.1. CoAP Option Number Registration . . . . .	35
9. Acknowledgments . . . . .	36
10. References . . . . .	36
10.1. Normative References . . . . .	36
10.2. Informative References . . . . .	38
Appendix A. Design Justification . . . . .	40
Appendix B. Roles and Responsibilities -- a Checklist . . . . .	41
Appendix C. Optimizations . . . . .	44
Appendix D. CoAP and CBOR profiles for OAuth 2.0 . . . . .	45
D.1. Profile for Token resource . . . . .	45
D.1.1. Token Request . . . . .	46
D.1.2. Token Response . . . . .	47

D.2. CoAP Profile for OAuth Introspection . . . . .	48
D.2.1. Introspection Request . . . . .	48
D.2.2. Introspection Response . . . . .	49
Appendix E. Document Updates . . . . .	51
E.1. Version -00 to -01 . . . . .	51
Authors' Addresses . . . . .	52

## 1. Introduction

Authorization is the process for granting approval to an entity to access a resource [RFC4949]. 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 requesting client, for 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 access-control and authorization for their Internet of Things (IoT) devices in the same style and this desire will increase with the number of exposed services and capabilities provided by applications hosted on the IoT devices. The IoT devices may be constrained in various ways including processing, memory, code-size, 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 devices connected to a continuous power supply 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 constraints. 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 smaller 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. Transport layer security can be provided either by DTLS 1.2 [RFC6347] or 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 nevertheless takes all these aspects into account and allows several different deployment variants to co-exist rather than mandating a one-size-fits-all solution. We believe this is important to cover the wide range of possible interworking use cases and the different requirements from a security point of view. Once IoT deployments mature, popular deployment variants will be documented in form of 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 asymmetric cryptography. The appropriate choice of security depends on the constraints of the IoT devices as well as on the security requirements of the use case.

#### Symmetric PoP key:

The AS generates a random symmetric PoP key, 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 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 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. As the client could be a constrained device as well, this memo uses CBOR encoded messages defined in Appendix D 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 [I-D.wahlstroem-ace-cbor-web-token] an equivalent format using CBOR encoding (CWT) has been defined.

#### Introspection:

Introspection is a method for a resource server to query the authorization server for the active state and content of a received access token. This is particularly useful in those cases where the authorization decisions are very dynamic and/or where the received access token itself is a reference rather than a self-contained token. More information about introspection in OAuth 2.0 can be found in [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 reordering and loss of packets can occur. A security solution need to take the latter aspects into account.

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

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [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. While OSCOAP integrity protects specific CoAP message meta-data like request/response code, and binds a response to a specific request, OSCON protects only payload/content, therefore 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 make a request to `"/.well-known/core"` to obtain information about available resources, which are returned in a standardized format as described in [RFC6690].



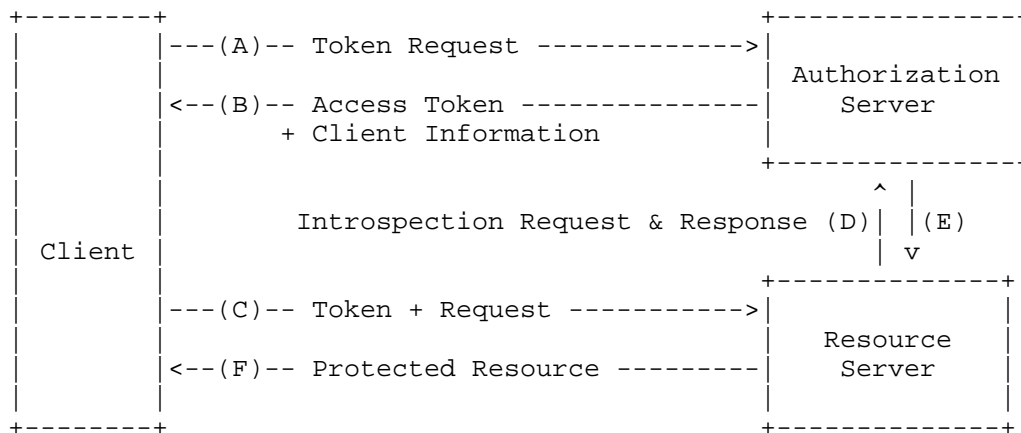


Figure 1: Overview of the basic protocol flow

## 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. The RS authenticates the client via the PoP token; and the client authenticates the RS 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 5.3) , which conveys authorization information to the RS that may be used by the client 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. Profiling for JSON Web Tokens (JWT) is provided in [I-D.wahlstroem-ace-cbor-web-token].

### 5.1. Client Information

OAuth 2.0 using bearer tokens, as described in [RFC6749] and in [RFC6750], 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 enables the AS to inform the client on how to communicate securely with a RS and that allows the client to indicate communication security preferences to the AS.

In the OAuth memo defining the key distribution for proof-of-possession (PoP) tokens [I-D.ietf-oauth-pop-key-distribution], the authors suggest to use Uri-query parameters in order to submit the parameters of the client's token request. To avoid large headers if the client uses CoAP to communicate with the AS, this memo specifies the following alternative for submitting client request parameters to the AS: The client encodes the parameters of it's request as a CBOR map and submits that map as the payload of the client request. The Content-format MUST be application/cbor in that case.

The OAuth memo further specifies that the AS SHALL use a JSON structure in the payload of the response to encode the response parameters. These parameters include the access token, destined for the RS and additional information for the client, such as e.g. the PoP key. We call this information "client information". If the client is using CoAP to communicate with the AS the AS SHOULD use CBOR instead of JSON for encoding it's response. The client can explicitly request this encoding by using the CoAP Accept option.

If the channel between client and AS is not secure, the whole messages from client to AS and vice-versa MUST be wrapped in JWES [RFC7516] or COSE\_Encrypted structures [I-D.ietf-cose-msg].

The client may be a constrained device and could therefore be limited in the communication security protocols it supports. It can

therefore signal to the AS which protocols it can support for securing their mutual communication. This is done by using the "csp" parameter defined below in the Token Request message sent to the AS.

Note that The OAuth key distribution specification [I-D.ietf-oauth-pop-key-distribution] describes in section 6 how the client can request specific types of keys (symmetric vs. asymmetric) and proof-of-possession algorithms in the PoP token request.

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", "ObjectSecurity" with the encodings specified in Figure 2. Depending on the value different additional parameters become mandatory.

Value	Major Type	Key
0	0	TLS
1	0	DTLS
2	0	ObjectSecurity

Figure 2: Table of 'csp' parameter value encodings for Client Information.

CoAP specifies three security modes of DTLS: PreSharedKey, RawPublicKey and Certificate. The same modes may be used with TLS. The client is to infer from the type of key provided, which (D)TLS mode the RS supports as follows.

If PreSharedKey mode is used, the AS MUST provide the client with the pre-shared key to be used with the RS. This key MUST be the same as the PoP key (i.e. a symmetric key as in section 4 of [I-D.ietf-oauth-pop-key-distribution]).

The client MUST use the PoP key as DTLS pre-shared key. The client MUST furthermore use the "kid" parameter provided as part of the JWK/COSE\_Key as the psk\_identity in the DTLS handshake [RFC4279].

If RawPublicKey mode is used, the AS MUST provide the client with the RS's raw public key using the "rpki" parameter defined in the

following. This parameter MUST contain a JWK or a COSE\_Key. The client MUST provide a raw public key to the AS, and the AS MUST use this key as PoP key in the token. The token MUST thus use asymmetric keys for the proof-of-possession.

In order to get the proof-of-possession a RS configured to use this mode together with PoP tokens MUST require client authentication in the DTLS handshake. The client MUST use the raw public key bound to the PoP token for client authentication in DTLS.

TLS or DTLS with certificates MAY make use of pre-established trust anchors or MAY be configured more tightly with additional client information parameters, such as x5c, x5t, or x5t#S256. An overview of these parameters is given below.

For when communication security is based on certificates this attribute can be used to define the server certificate or CA certificate. Semantics for this attribute is defined by [RFC7517] or COSE\_Key [I-D.ietf-cose-msg].

For when communication security is based on certificates this attribute can be used to define the specific server certificate to expect or the CA certificate. Semantics for this attribute is defined by JWK/COSE\_Key.

To use object security (such as OSCOAP and OSCON) requires security context to be established, which can be provisioned with PoP token and client information, or derived from that information. Object security specifications designed to be used with this protocol MUST specify the parameters that an AS has to provide to the client in order to set up the necessary security context.

The RS may support different ways of receiving the access token from the client (see Section 5.3 and Appendix C). The AS MAY signal the required method for access token transfer in the client information by using the "tktr" (token transport) parameter using the values defined in table Figure 3. If no "tktn" parameter is present, the client MUST use the default Authorization Information resource as specified in Section 5.3.

Value	Major Type	Key
0	0	POST to /authz-info
1	0	RFC 4680
2	0	CoAP option "Ref-Token"

Figure 3: Table of 'tktn' parameter value encodings for Client Information.

Table Figure 4 summarizes the additional parameters defined here for use by the client or the AS in the PoP token request protocol.

Parameter	Used by	Description
csp	client or AS	Communication security protocol
rpk	AS	RS's raw public key
x5c	AS	RS's X.509 certificate chain
x5t	AS	RS's SHA-1 cert thumb print
x5t#S256	AS	RS's SHA-256 cert thumb print
tktn	AS	Mode of token transfer C -> RS

Figure 4: Table of additional parameters defined for the PoP protocol.

## 5.2. CoAP Access-Token Option

OAuth 2.0 access tokens are usually transferred as authorization header. CoAP has no authorization header equivalence. This document therefor register the option Access-Token. The Access-Token option is an alternative for transferring the access token when it is smaller then 255 bytes. If token is larger the 255 bytes lager authorization information resources MUST at the RS be user when CoAP.

## 5.3. 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 than 255 bytes it should not be sent as a CoAP option.

For conveying authorization information to the RS a new resource is introduced 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.

The RS needs to store the PoP token for when later authorizing requests from the client. The RS is not mandated to be able to manage multiple client at once. how the RS manages clients is out of scope for this specification.

#### 5.3.1. Authorization Information Request

The client makes a POST request to the authorization information resource by sending its PoP token as request data.

Client MUST send the Content-Format option indicate token format

#### 5.3.2. Authorization Information Response

The RS MUST responde to a requests to the authorization information resource. The response MUST match CoAP response codes according to success or error response section

##### 5.3.2.1. Success Response

Successful requests MUST be answered with 2.01 Created to indicate that a "session" for the PoP Token has been created. No location path is required to be returned.

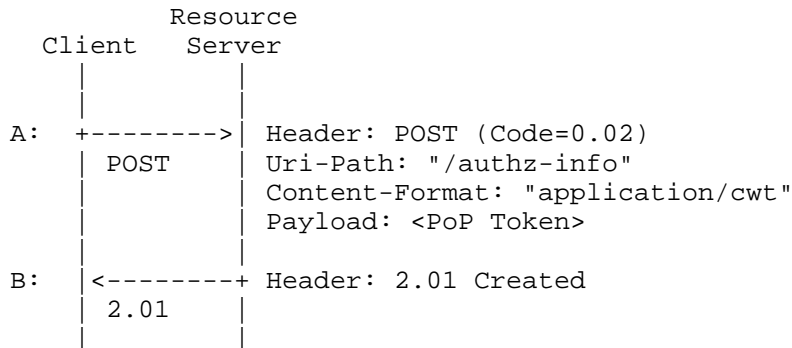


Figure 5: Authorization Information Resource Success Response

##### 5.3.2.2. Error Response

The resource server MUST user appropriate CoAP response code to convey the error to the Client. For request that are not valid, e.g. unknown Content-Format, 4.00 Bad Request MUST be returned. If token

is not valid, e.g. wrong audience, the RS MUST return 4.01 Unauthorized.

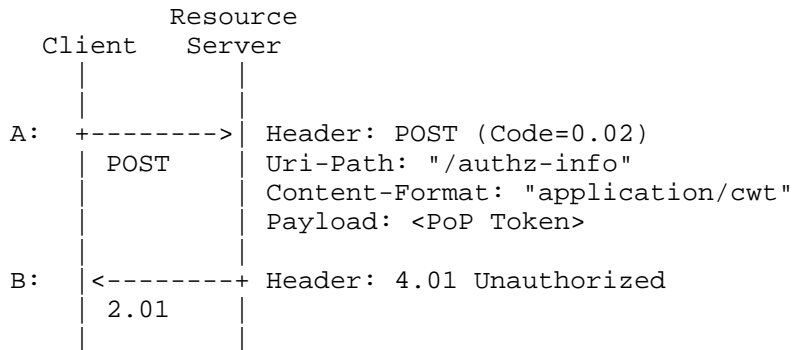


Figure 6: Authorization Information Resource Error Response

#### 5.4. 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). Other formats may be mandated by specific applications or requirements (e.g. specifying local conditions on access).

#### 5.5. 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 D defines a CBOR-based format for sending parameters to the /token resource.

#### 5.6. Token Expiration

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



- o The token is a CWT/JWT and includes a 'exp' claim and possibly the 'nbf' claim. The RS verifies these by comparing them to values from its internal clock as defined in [RFC7519]. In this case the RS must have a real time chip (RTC) or some other way of reliably measuring time.
- o The RS verifies the validity of the token by performing an introspection request as specified in Appendix D.2. This requires the RS to have a reliable network connection to the AS and to be able to handle two secure sessions in parallel (C to RS and AS to RS).
- o The RS and the AS both store a sequence number linked to their common security association. The AS increments this number for each access token it issues and includes it in the access token, which is a CWT/JWT. The RS keeps track of the most recently received sequence number, and only accepts tokens as valid, that are in a certain range around this number. This method does only require the RS to keep track of the sequence number. The method does not provide timely expiration, but it makes sure that older tokens cease to be valid after a specified number of newer ones got issued. For a constrained RS with no network connectivity and no means of reliably measuring time, this is the best that can be achieved.

## 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 7.

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 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.

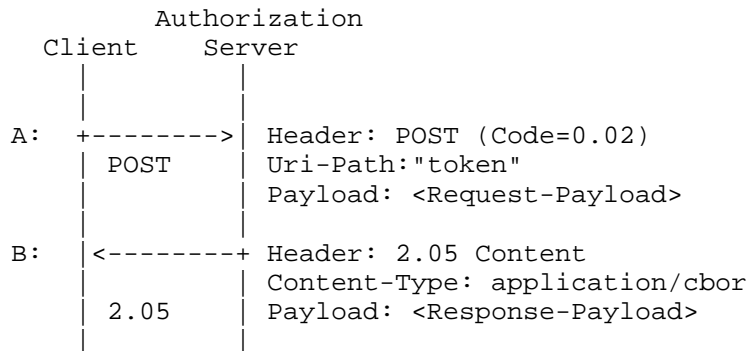


Figure 7: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 8.

```
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 8: Request and Response Payload Details.

The content of the "key" parameter and the access token are shown in Figure 9 and Figure 10.

```
{
  "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
  "kty" : "EC",
  "crv" : "P-256",
  "x" : b64'MKBCTNIcKUSDiillySs3526iDZ8AiTo7Tu6KPAqv7D4',
  "y" : b64'4Et16SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
}
```

Figure 9: Public Key of the RS.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "cnf" : {
    "jwk" : {
      "kid" : b64'1Bg8vub9tLelgHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f83OJ3D2xF1Bg8vub9tLelgHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}
```

Figure 10: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 11 - Figure 12.

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.

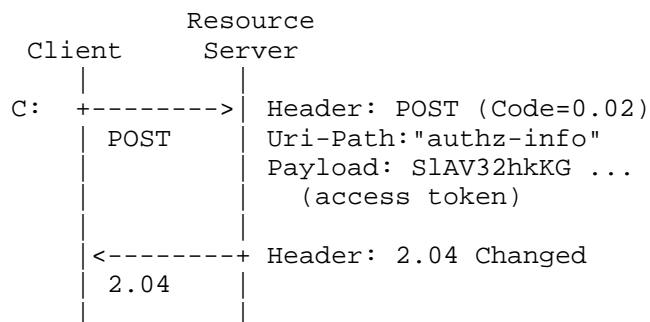


Figure 11: Access Token provisioning to RS

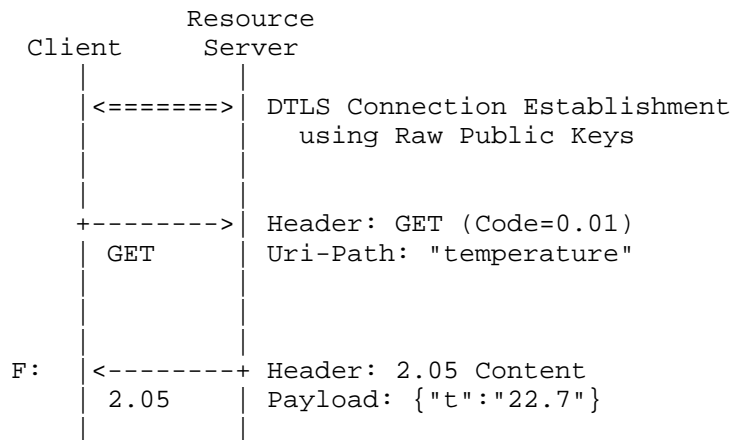


Figure 12: Resource Request and Response protected by DTLS.

## 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 (air-conditioning control unit), an offline resource server (temperature sensor) and an authorization server. The message exchanges A and B are shown in Figure 13.

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 wants 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.

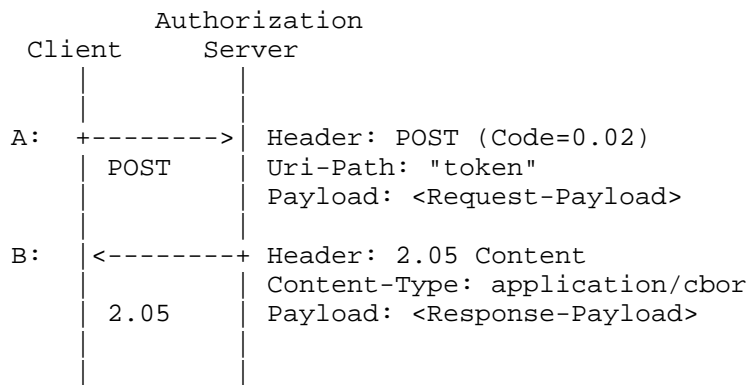


Figure 13: Token Request and Response

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

```

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 14: Request and Response Payload for RS offline

Figure 15 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_FzUA5XKMYoVHyzzff5oRJxl-IXRtztJ6uE'
}

```

Figure 15: Access Token and symmetric key from the Response-Payload

Message exchanges C and F are shown in Figure 16 and Figure 17.

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.

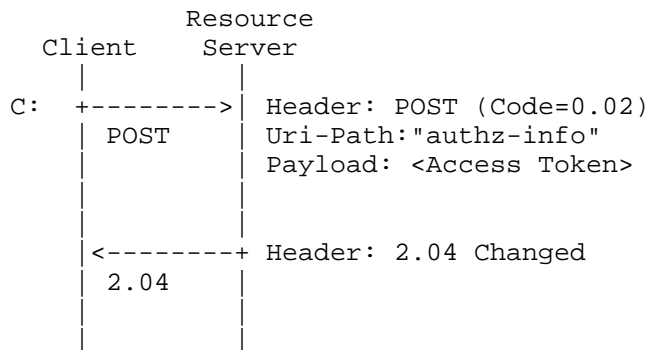


Figure 16: Access Token provisioning to RS

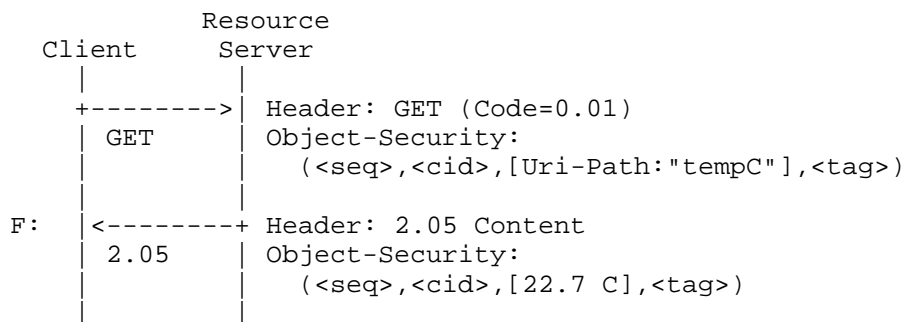


Figure 17: Resource request and response protected by OSCOAP

In Figure 17 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 18.

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.

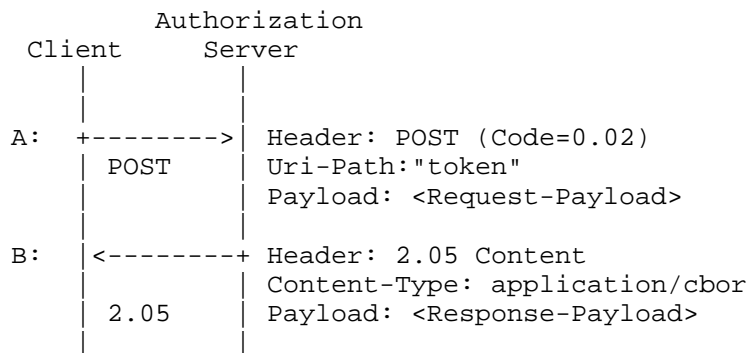


Figure 18: Token Request and Response using Client Credentials.

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 19.

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 19: 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).

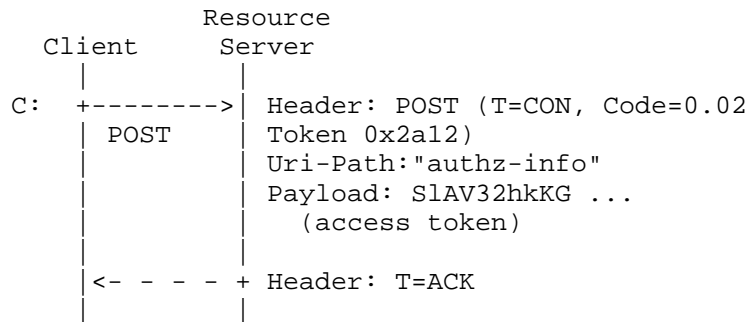


Figure 20: Access Token provisioning to RS

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.

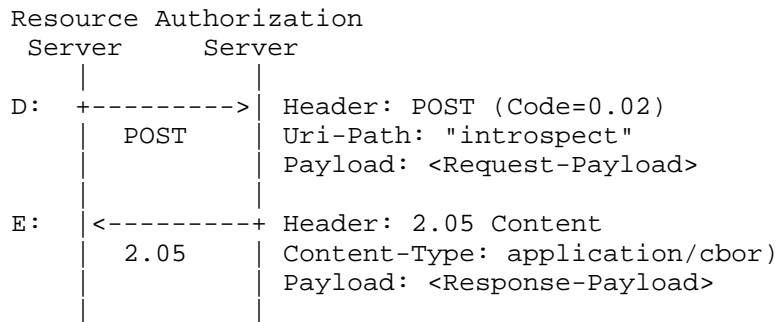


Figure 21: Token Introspection for C offline

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

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 22: 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.

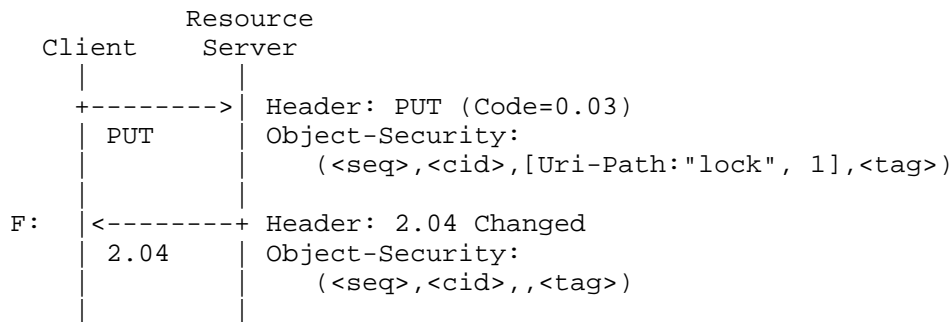


Figure 23: Resource request and response protected by OSCOAP

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 [I-D.wahlstroem-ace-cbor-web-token] 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 24.

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.

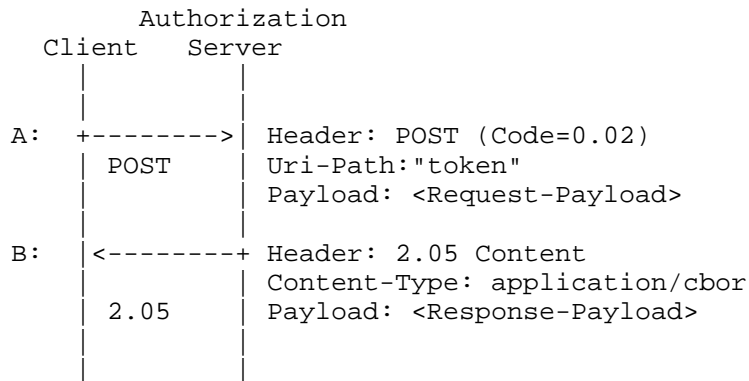


Figure 24: Token Request and Response

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

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 25: Request and Response Payload for RS severely constrained

The content of the "key" parameter is shown in Figure 26.

```
key :
{
  "alg" : "AES_128_CTR_ECDSA",
  "kid" : b64'c29tZSBvdGhlciBrZXkgaWQ';
  "k" : b64'ZoRSOrFzN_FzUA5XKMYoVHyzzff5oRJxl-IXRtztJ6uE',
  "crv" : "P-256",
  "x" : b64'MKBCtNICKUSDiillySs3526iDZ8AiTo7Tu6KPAqv7D4',
  "y" : b64'4Et16SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
}
```

Figure 26: 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 27.

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 27.

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.



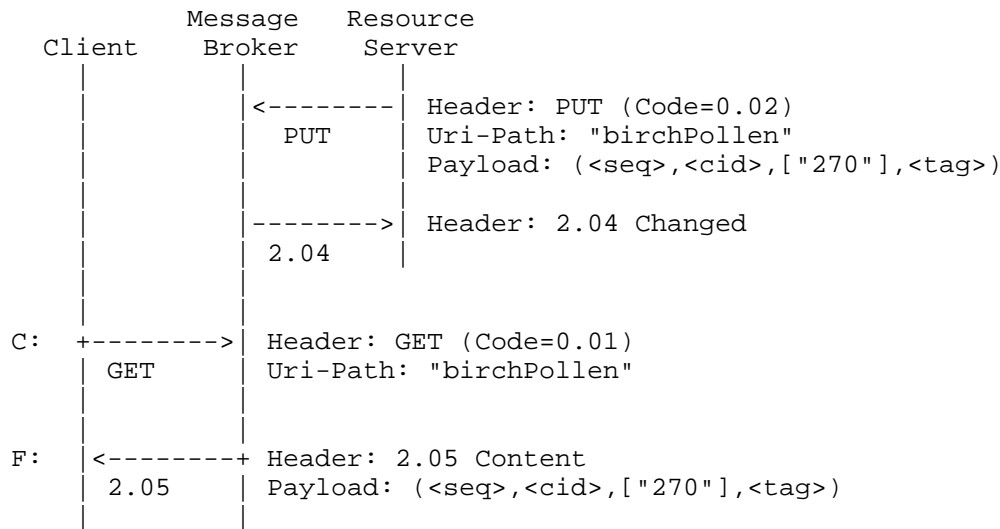


Figure 27: Sensor measurement protected by COSE

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 26, 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

FIXME: Add registry over 'csp' values from Figure 2

FIXME: Add registry of 'rpk' parameter from section 5.1

FIXME: Add registry of 'tktn' values from Figure 3

### 8.1. CoAP Option Number Registration

This section registers the "Access-Token" CoAP Option Number [RFC2046] in "CoRE Parameters" sub-registry "CoAP Option Numbers" in the manner described in [RFC7252].

Name

Access-Token

Number

TBD

## Reference

[draft-ietf-ace-oauth-authz]

## Meaning in Request

Contains an Access Token according to [draft-ietf-ace-oauth-authz] containing access permissions of the client.

## Meaning in Response

Not used in response

## Safe-to-Forward

TBD

## Format

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

## Length

Less than 255 bytes

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

## 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-03 (work in progress), July 2015.

[I-D.ietf-cose-msg]

Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-10 (work in progress), February 2016.

- [I-D.ietf-oauth-introspection]  
Richer, J., "OAuth 2.0 Token Introspection", draft-ietf-oauth-introspection-11 (work in progress), July 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-07 (work in progress), December 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-02 (work in progress), October 2015.
- [I-D.selander-ace-object-security]  
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-03 (work in progress), October 2015.
- [I-D.wahlstroem-ace-cbor-web-token]  
Wahlstroem, E., Jones, M., and H. Tschofenig, "CBOR Web Token (CWT)", draft-wahlstroem-ace-cbor-web-token-00 (work in progress), December 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, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <<http://www.rfc-editor.org/info/rfc4279>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.

## 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-02 (work in progress), October 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.seitz-ace-core-authz] Seitz, L., Selander, G., and M. Vucinic, "Authorization for Constrained RESTful Environments", draft-seitz-ace-core-authz-00 (work in progress), June 2015.
- [I-D.somaraju-ace-multicast] Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-01 (work in progress), January 2016.
- [RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, DOI 10.17487/RFC4680, October 2006, <<http://www.rfc-editor.org/info/rfc4680>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<http://www.rfc-editor.org/info/rfc4949>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<http://www.rfc-editor.org/info/rfc6750>>.
- [RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<http://www.rfc-editor.org/info/rfc6819>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.

## Appendix A. Design Justification

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

Common IoT constraints are:

### Low Power Radio:

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

### Low CPU Speed:

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

### Small Amount of Memory:

Microcontrollers embedded in IoT devices are often equipped with small amount of RAM and flash memory, which places limitations what kind of processing can be performed and how much code can be put on those devices. To reduce code size fewer and smaller protocol implementations can be put on the firmware of such a device. In this case, CoAP may be used instead of HTTP, symmetric

key cryptography instead of public key cryptography, and CBOR instead of JSON. Authentication and key establishment protocol, e.g. the DTLS handshake, in comparison with assisted key establishment also has an impact on memory and code.

#### User Interface Limitations:

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

#### Communication Constraints:

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

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

## Appendix B. Roles and Responsibilities -- a Checklist

### Resource Owner

- \* Make sure that the RS is registered at the AS.
- \* Make sure that clients can discover the AS which is in charge of the RS.
- \* Make sure that the AS has the necessary, up-to-date, access control policies for the RS.



## Requesting Party

- \* Make sure that the client is provisioned the necessary credentials to authenticate to the AS.
- \* Make sure that the client is configured to follow the security requirements of the Requesting Party, when issuing requests (e.g. minimum communication security requirements, trust anchors).
- \* Register the client at the AS.

## Authorization Server

- \* Register RS and manage corresponding security contexts.
- \* Register clients and including authentication credentials.
- \* Allow Resource Owners to configure and update access control policies related to their registered RS'
- \* Expose a service that allows clients to request tokens.
- \* Authenticate clients that wishes to request a token.
- \* Process a token requests against the authorization policies configured for the RS.
- \* Expose a service that allows RS's to submit token introspection requests.
- \* Authenticate RS's that wishes to get an introspection response.
- \* Process token introspection requests.
- \* Optionally: Handle token revocation.

## Client

- \* Discover the AS in charge of the RS that is to be targeted with a request.
- \* Submit the token request (A).
  - + Authenticate towards the AS.
  - + Specify which RS, which resource(s), and which action(s) the request(s) will target.

- + Specify preferences for communication security
- + If raw public key (rpk) or certificate is used, make sure the AS has the right rpk or certificate for this client.
- \* Process the access token and client information (B)
  - + Check that the token has the right format (e.g. CWT).
  - + Check that the client information provides the necessary security parameters (e.g. PoP key, information on communication security protocols supported by the RS).
- \* Send the token and request to the RS (C)
  - + Authenticate towards the RS (this could coincide with the proof of possession process).
  - + Transmit the token as specified by the AS (default is to an authorization information resource, alternative options are as a CoAP option or in the DTLS handshake).
  - + Perform the proof-of-possession procedure as specified for the type of used token (this may already have been taken care of through the authentication procedure).
- \* Process the RS response (F) requirements of the Requesting Party, when issuing requests (e.g. minimum communication security requirements, trust anchors).
- \* Register the client at the AS.

#### Resource Server

- \* Expose a way to submit access tokens.
- \* Process an access token.
  - + Verify the token is from the right AS.
  - + Verify that the token applies to this RS.
  - + Check that the token has not expired (if the token provides expiration information).
  - + Check the token's integrity.

- + Store the token so that it can be retrieved in the context of a matching request.
- \* Process a request.
  - + Set up communication security with the client.
  - + Authenticate the client.
  - + Match the client against existing tokens.
  - + Check that tokens belonging to the client actually authorize the requested action.
  - + Optionally: Check that the matching tokens are still valid (if this is possible).
- \* Send a response following the agreed upon communication security.

## Appendix C. 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 provisioning 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 beneficial to utilize 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 self-contained token, 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 28.

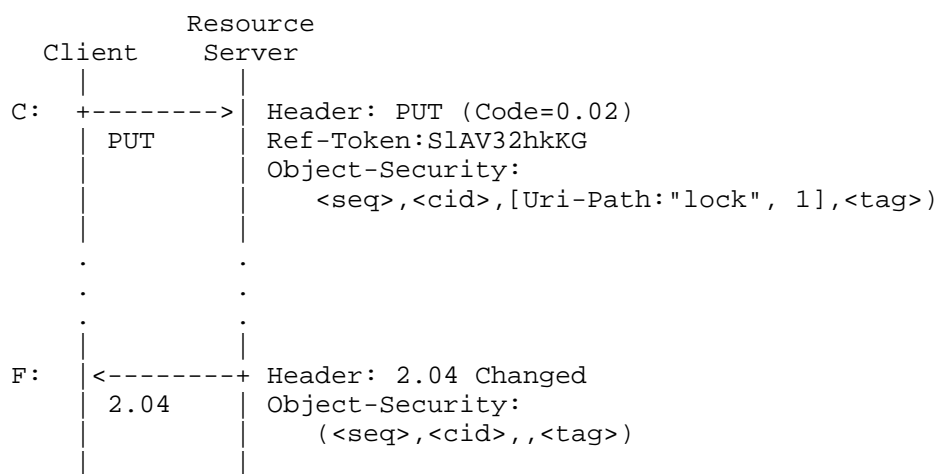


Figure 28: Reference Token in CoAP Option

## Appendix D. 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.

## D.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.

## D.1.1.1. 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 [I-D.wahlstroem-ace-cbor-web-token].

**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 29: CBOR mappings used in token requests

## D.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 30: CBOR mappings used in token responses

## D.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.

### D.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 31: CBOR Mappings to Token Introspection Request Parameters.

#### D.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 [I-D.wahlstroem-ace-cbor-web-token].

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 [I-D.wahlstroem-ace-cbor-web-token].

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 [I-D.wahlstroem-ace-cbor-web-token].

sub:

OPTIONAL. Subject of the token, as defined in CWT [I-D.wahlstroem-ace-cbor-web-token]. 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 [I-D.wahlstroem-ace-cbor-web-token].

iss:

OPTIONAL. String representing the issuer of this token, as defined in CWT [I-D.wahlstroem-ace-cbor-web-token].

cti:

OPTIONAL. String identifier for the token, as defined in CWT [I-D.wahlstroem-ace-cbor-web-token]

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 32: CBOR Mappings to Token Introspection Response Parameters.

## Appendix E. Document Updates

### E.1. Version -00 to -01

- o Changed 5.1. from "Communication Security Protocol" to "Client Information".
- o Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP token request profile for IoT.
  - \* Allow the client to indicate preferences for the communication security protocol.
  - \* Defined the term "Client Information" for the additional information returned to the client in addition to the access token.

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

#### Authors' Addresses

Ludwig Seitz  
SICS  
Scheelevaegen 17  
Lund 223 70  
SWEDEN

Email: ludwig@sics.se

Goeran Selander  
Ericsson  
Farogegatan 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.erdman@nexusgroup.com

Hannes Tschofenig  
ARM Ltd.  
Hall in Tirol 6060  
Austria

Email: Hannes.Tschofenig@arm.com

ace  
Internet-Draft  
Intended status: Standards Track  
Expires: May 4, 2017

A. Somaraju  
Tridonic GmbH & Co KG  
S. Kumar  
Philips Research  
H. Tschofenig  
ARM Ltd.  
W. Werner  
Werner Management Services e.U.  
October 31, 2016

Security for Low-Latency Group Communication  
draft-somaraju-ace-multicast-02.txt

Abstract

Some Internet of Things application domains require secure group communication. This draft describes procedures for authorization, key management, and securing group messages. We specify the usage of object security at the application layer for group communication and assume that CoAP is used as the application layer protocol. The architecture allows the usage of symmetric and asymmetric keys to secure the group messages. The asymmetric key solution provides the ability to uniquely authenticate the source of all group messages and this is the recommended architecture for most applications. However, some applications have strict requirements on latency for group communication (e.g. in non-emergency lighting applications) and it may not always be feasible to use the secure source authenticated architecture. In such applications we recommend the use of dynamically generated symmetric group keys to secure group communications.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

## Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	3
3. Architecture - Group Authentication . . . . .	5
3.1. Assumptions . . . . .	8
3.2. AT-KDC Access Tokens . . . . .	9
3.3. AT-R Access Tokens . . . . .	9
3.4. Multicast Message Content . . . . .	10
3.5. Receiver Algorithm . . . . .	11
3.6. Sender Algorithm . . . . .	12
4. Architecture - source authentication . . . . .	14
4.1. Assumptions . . . . .	16
4.2. AT-R Access Tokens . . . . .	17
4.3. Multicast Message Content . . . . .	17
4.4. Receiver Algorithm . . . . .	18
4.5. Sender Algorithm . . . . .	19
5. Security Considerations . . . . .	20
5.1. Applicability statement . . . . .	20
5.2. Token Verification . . . . .	21
5.3. Token Revocation . . . . .	21
5.4. Time . . . . .	22
6. Operational Considerations . . . . .	22
6.1. Persistence of State Information . . . . .	22
6.2. Provisioning in Small Networks . . . . .	23
6.3. Client IDs . . . . .	23
6.4. Application Groups vs. Security Groups . . . . .	23
6.5. Lost/Stolen Device . . . . .	23
7. Acknowledgements . . . . .	24
8. IANA Considerations . . . . .	24
9. References . . . . .	24
9.1. Normative References . . . . .	24
9.2. Informative References . . . . .	25

Appendix A. Access Levels . . . . .	25
Authors' Addresses . . . . .	26

## 1. Introduction

There are low latency group communication use cases that require securing communication between a sender, or a group of senders, and a group of receivers. In the lighting use case, a set of lighting nodes (e.g., luminaires, wall-switches, sensors) are grouped together into a single "Application Group" and the following three requirements need to be addressed:

1. Only authorized members of the application group must be able to read and process messages.
2. Receivers of group messages must be able to verify the integrity of received messages as being generated within the group.
3. Message communication and processing must happen with a low latency and in synchronous manner.

This document discusses a group communication security solution that satisfies these three requirements. As discussed in Section 4, we recommend the usage of an asymmetric key solution that allows unique source authentication of all group messages. However, in situations where the low latency requirements can not be met (e.g. in non-emergency lighting applications), the alternative architecture discussed in Section 3 based on symmetric keys is recommended.

## 2. Terminology

This document uses the following terms from [I-D.ietf-ace-actors]: Authorization Server, Resource Owner, Client, Resource Server. The terms 'sender' and 'receiver' refer to the application layer messaging used for lighting control; other communication interactions with the supporting infrastructure uses unicast messaging.

When nodes are combined into groups there are different layers of those groups with unique characteristics. For clarity we introduce terminology for three different groups:

### Application Group:

An application group consists of the set of all nodes that have been configured to respond to a single application layer request. For example, a wall mounted switch and a set of luminaires in a single room might belong to a single group and the switch may be used to turn on/off all the luminaires in the group simultaneously

with a single button press. In the remainder of this document we will use GID to identify an application group.

#### Multicast Group:

A multicast group consists of the set of all nodes that subscribe to the same multicast IP address.

#### Security Group:

A security group consists of the set of all nodes that have been provisioned with the same keying material. All the nodes within a security group share a security association or a sequence of security associations wherein a single association specifies the keying material, algorithm-specific information, lifetime and a key ID.

#### Source-authenticated Security Group:

A source-authenticated security group consists of the set of receiver nodes that have been provisioned with the public verification keying material of all the sender nodes and the set of sender nodes that are provisioned with their unique private signing keying material. All the nodes within a source-authenticated security group share a security association or a sequence of security associations wherein a single association specifies the the public or private keying material, algorithm-specific information, lifetime and a key ID.

Typically, the four groups might not coincide due to the memory constraints on the devices and also security considerations. For instance, in a small room with windows, we may have three application groups: "room group", "luminaires close to the window group" and "luminaires far from the window group". However, we may choose to use only one multicast group for all devices in the room and one security group for all the devices in the room. Note that every application group belongs to a unique security group. However, the converse is not always true. This implies that the application group ID maybe used to determine the associated security group but not vice versa.

The fact that security groups may not coincide with application groups implies that

- (1) an application must be able to specify which resources on a resource server are accessible by a client that has access to the group key, and



(2) a method is required to associate the group key to the application group(s) for which the group key may be used.

In this document we provide fields that may be used to specify the "scope of the key" and "application groups for which the key may be used". A commissioner has a lot of flexibility to assign nodes to multicast groups and to security groups while the application groups will be determined by the semantics of the application itself. The exact partitioning of the nodes into security and multicast groups is therefore deployment specific.

### 3. Architecture - Group Authentication

Each node in a lighting application group might be a sender, a receiver or both sender and receiver (even though in Figure 1, we show nodes that are only senders or only receivers for clarity). The low latency requirement implies that most of the communication between senders and receivers of application layer messages is done using multicast IP. On some occasions, a sender in a group will be required to send unicast messages to unique receivers within the same group and these unicast messages also need communication security.

Two logical entities are introduced and they have the following function:

**Key Distribution Center (KDC):** This logical entity is responsible for generating symmetric keys and distributing them to the nodes authorized to receive them. The KDC ensures that nodes belonging to the same security group receive the same key and that the keys are renewed based on certain events, such as key expiry or change in group membership.

**Authorization Server (AS):** This logical entity stores authorization information about devices, meta-data about them, and their roles in the network. For example, a luminaire is associated with different groups, and may have meta-data about its location in a building.

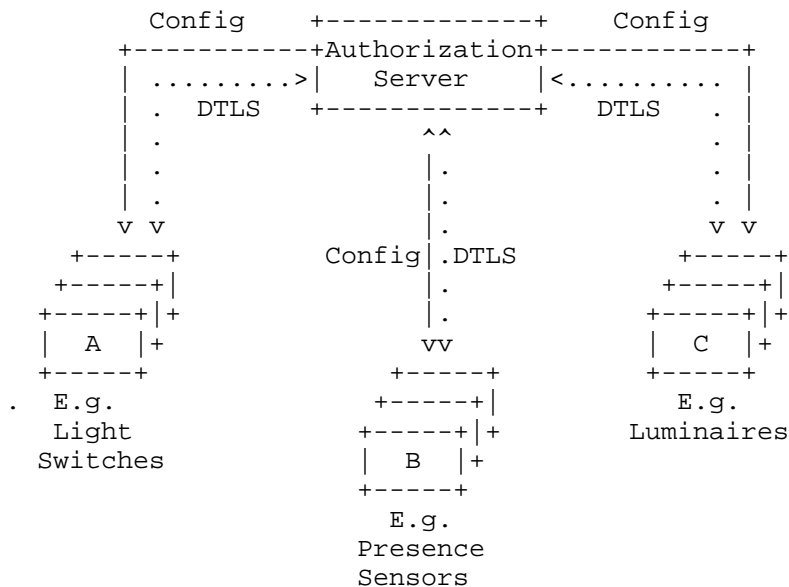
Note that we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 1 and Figure 2 provide an architectural overview. The dotted lines illustrate the use of unicast DTLS messages for securing the message exchange between all involved parties. The secured group messages between senders and receivers are indicated using lines with

star/asterisk characters. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) which are to be defined.

Figure 1 illustrates the information flow between an authorization server and the nodes participating in the lighting network, which includes all nodes that exchange lighting application messages. This step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control lights, may require an authorization server that dynamically executes access control decisions.

Figure 1 shows the commissioning phase where the nodes obtain configuration information, which includes the AT-KDC. The AT-KDC is an access token and includes authorization claims for consumption by the key distribution center. We use the access token terminology from [RFC6749]. The AT-KDC in this architecture may be a bearer token or a proof-of-possession (PoP) token. The bearer token concept is described in [RFC6750] and the PoP token concept is explained in [I-D.ietf-oauth-pop-architecture]. The AT-KDC is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-KDC is protected against modifications using a digital signature or a message authentication code. It is verified in Figure 2 by the KDC.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-KDC) and other meta-data.

Figure 1: Architecture: Commissioning Phase.

In the simplified message exchange shown in Figure 2 a sender requests a security group key and the access token for use with the receivers (called AT-R). The request contains information about the resource it wants to access, such as the application group and other resource-specific information, if applicable, and the previously obtained AT-KDC access token. Once the sender has successfully obtained the requested information it starts communicating with receivers in that group using group messages. The symmetric key obtained from the KDC is used to secure the groups messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary group key to verify the incoming messages and to determine what resource the requestor is authorized to access. Both pieces of information can be found in the AT-R access token.

Group messages need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using

a keyed message digest in combination with the group key. The use of symmetric keys is envisioned in this specification due to latency requirements. For unicast messaging between the group members and the AS or KDC, we assume the use of DTLS for transport security. However, the use of TLS, and application layer security is possible but is outside the scope of this document.

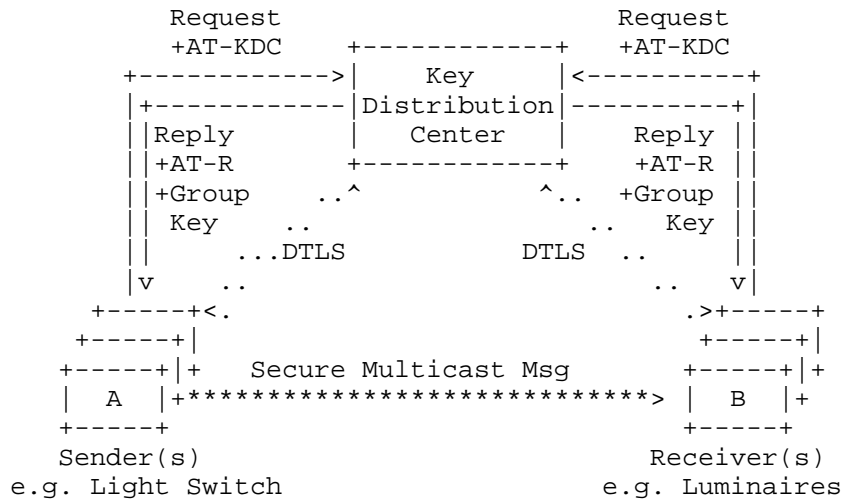


Figure 2: Architecture: Group Key Distribution Phase.

### 3.1. Assumptions

1. The AT-KDC is a manifestation of the authorization granted to a specific client (or user running a client). The AT-KDC is longer-lived and can be used to request multiple AT-Rs.
2. Each AT-R is valid for use with one or multiple application groups.
3. The AS and the KDC logical roles may reside in different physical entities.
4. The AT-KDC as well as the AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.
5. The AT-KDC token is opaque to the client. Data that is meant for processing by the client has to be conveyed to the client

separately. The AT-R token on the other hand is meant for consumption by the client.

6. The client requests AT-Rs for different application groups by including additional information in the request to the KDC for what application groups the AT-R(s) have to be requested. The KDC may return multiple AT-Rs in a single response (for performance reasons).
7. The AT-KDC and the AT-R are encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

### 3.2. AT-KDC Access Tokens

The AT-KDC contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Recipient/Audience: Indication to whom the AT-KDC was issued to. In this case, it is the KDC.
5. Client ID: Information about the client that was authenticated by the authorization server.
6. Issued at: Indicates date and time when the AT-KDC was created by the authorization server.

### 3.3. AT-R Access Tokens

Clients send the AT-KDC to the KDC in order to receive an AT-R.

The KDC MUST maintain a table consisting of scope values, which includes the application group id. These entries point to a sequence of security associations. A security association specifies the key material, algorithm-specific information, lifetime and a key ID and the key ID may be used to identify this security association.

The AS/KDC must guarantee the uniqueness of the client ids for its nodes. This may be accomplished by the AS/KDC assigning values to the nodes or by using information that is already unique per device (such as an EUI-64).

The KDC furthermore needs to be configured with information about the authorization servers it trusts. This may include a provisioned trust anchor store, or shared credentials (similar to a white list).

The KDC MUST generate new group keys after the validity period of the current group key expires.

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Security Group Key: Key to use for the group communication.
5. Algorithm: Used for secure group communication.
6. KID: Sequentially increasing ID of the key for the security group (the devices may store an older key to help with key rolling.)
7. Issued at: Indicates date and time when the AT-R was created by the KDC.

### 3.4. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of
  - \* Client ID (unencrypted, integrity protected): Every sender managed by a key distribution center MUST have a unique client ID.

- \* Sequence Number (unencrypted, integrity protected): Used for replay protection.
- \* An implicit IV that is either derived from the keys at the end-points or fixed to a certain value by standard (not sent in the message)

2. MAC (not integrity protected): For integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.
2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.
3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.
4. Key ID (unencrypted, integrity protected): Is used to select the current security association from the sequence of security associations identified by the application group id.
5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)
6. CoAP Payload (encrypted, integrity protected).

### 3.5. Receiver Algorithm

All receiving devices MUST maintain a table consisting of mappings of application group id, to a sequence of security associations.

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association.

The key id is used for situations where the group key is updated by the KDC (for example in situations where a device in a group is lost or stolen).

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver **MUST** silently discard an incoming message in the following cases:

- o Application Group ID lookup does not return any security association.
- o Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- o Integrity check fails.
- o Decryption fails.
- o Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association **MUST** be updated to the nonce value in the message.

### 3.6. Sender Algorithm

Figure 3 describes the algorithm for obtaining the necessary credentials to transmit a secure group message. When the sender wants to send a message to the application group, it checks if it has the respective group key. If no group key is available then it determines whether it has an access token for use with the KDC (i.e., AT-KDC). If no AT-KDC is found in the cache then it contacts the authorization server to obtain that AT-KDC. Note that this assumes



that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported. In the other case where the AT-KDC is already available the sender contacts the KDC to obtain a group key. If a group key is already available then the sender can transmit a secured message to the group immediately.

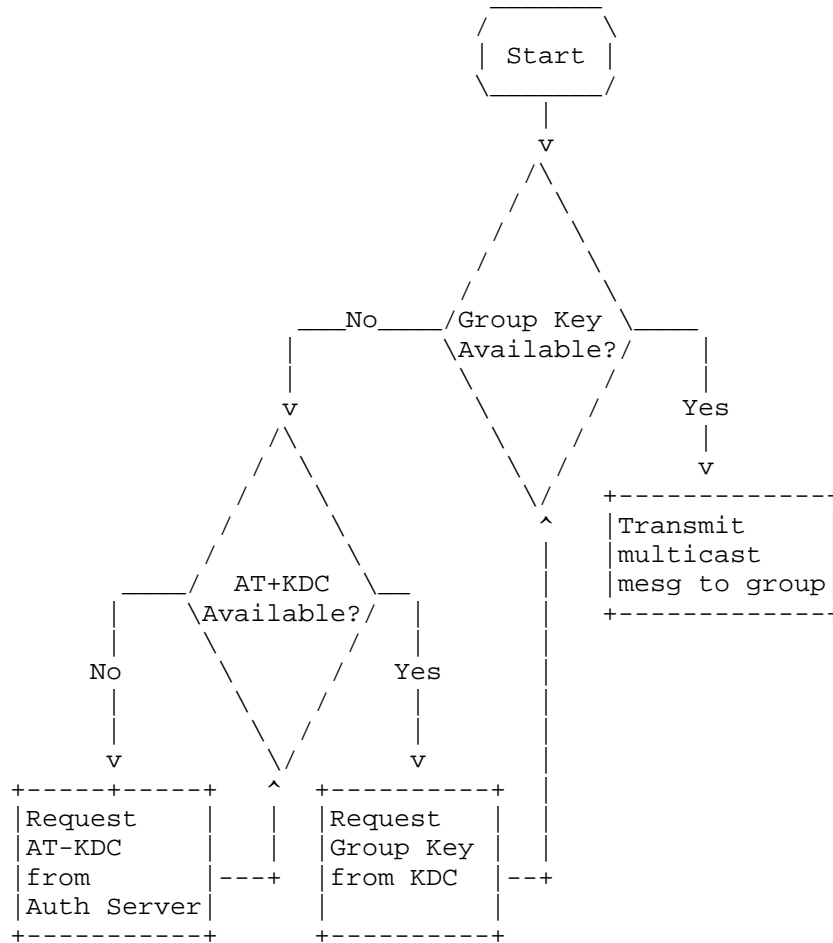


Figure 3: Steps to Transmit Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a group key; the sender is likely to be

pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

Group keys have a lifetime, which is configuration-dependent, but mechanisms need to be provided to update the group keys either via the sender asking for a group key renewal or via the KDC pushing new keys to senders and receivers. The lifetime can be based on time or on the number of transmitted messages.

#### 4. Architecture - source authentication

This section discusses the usage of asymmetric keys to achieve source authentication of group messages and is the recommend architecture for securing group messages. However, this solution may not meet the low latency requirement without adequate hardware support but still most of the group communication between senders and receivers of application layer messages is done using multicast IP.

Unlike the previous architecture, the current architecture requires only the Authorization Server (AS) logical entity as defined in the previous section.

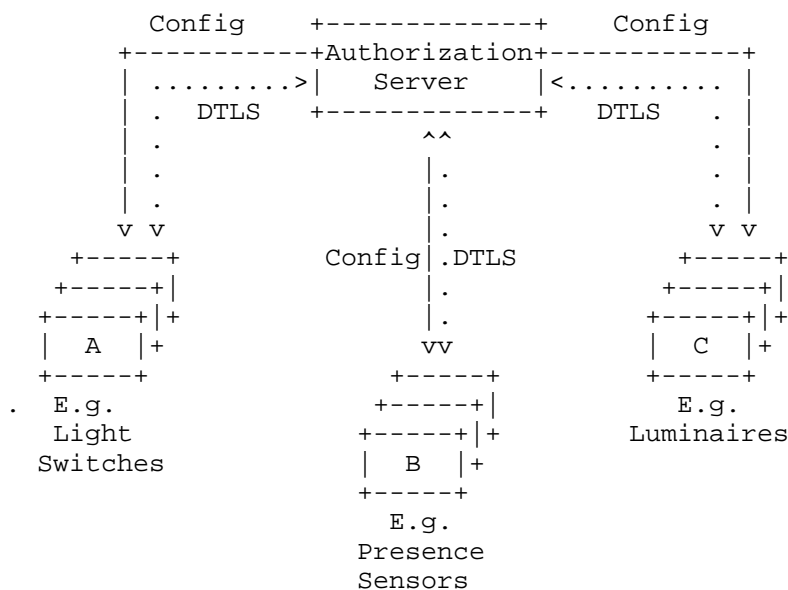
As in the previous case we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 4 and Figure 5 provide an architectural overview for the source authenticated case. The main differences from the previous case is that the AS provides directly the AT-R tokens. Further no KDC is required in this case since the senders and receivers can use their public-private key pair credentials to secure messages. The AS may provide authorization based on the pre-existing device credentials or issue new credentials to the devices. The security of the group messages is accomplished at the application level using small modification to OSCoAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) but based on public key signatures which are to be defined.

Figure 4 illustrates the information flow between an authorization server and the nodes participating in the source-authenticated group network. Like the previous case, this step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control

lights, may require an authorization server that dynamically executes access control decisions.

Figure 4 shows the commissioning phase where the nodes obtain configuration information, which includes directly the AT-R. The AT-R is an access token and includes authorization claims for consumption by the receivers. The AT-R may be a bearer token or a proof-of-possession (PoP) token. The AT-R is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-R is protected against modifications using a digital signature. It is verified in Figure 5 by the receivers.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-R) and other meta-data.

Figure 4: Architecture - Source-authenticated: Commissioning Phase.

In the simplified message exchange shown in Figure 5 a sender starts communicating with receivers in that source-authenticated group using public-key signed group messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary public verification key of the senders (or a root verification key if they are certified by the same authority) to verify the incoming messages and the public verification key of the AS to determine what resource the requestor is authorized to access. Both pieces of information can either be found in the AT-R access token or separately configured during the commissioning phase.

Source-authenticated Group messages also need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using a public-key signature. This may not achieve the latency requirements and used where source-authentication is more important. For unicast messaging between the group members and the AS, we assume the use of DTLS for transport security.

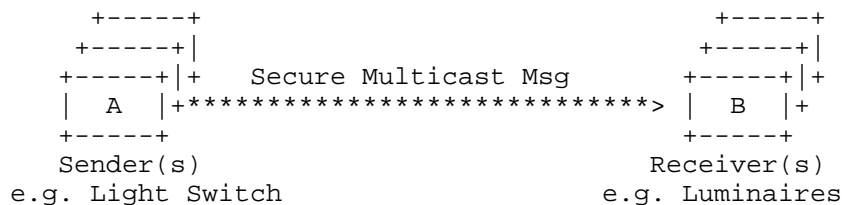


Figure 5: Architecture - Source-authenticated: Group communication.

#### 4.1. Assumptions

1. The AT-R is a manifestation of the authorization granted to a specific client (or user running a client). The AT-R is longer-lived and can be used directly for source-authenticated group communication until it is revoked or expired.
2. Each AT-R is valid for use with one or multiple application groups.
3. The AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.
4. The AT-R token is not opaque to the client and is meant for consumption by the client.
5. The client requests AT-Rs for different application groups by including additional information in the request to the AS for what application groups the AT-R(s) have to be requested. The AS

may return multiple AT-Rs in a single response (for performance reasons).

6. The AT-R is encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

#### 4.2. AT-R Access Tokens

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Recipient/Audience: Indication to whom the AT-R was issued to. In this case, it is the receivers.
5. Client ID: Information about the client that was authenticated by the authorization server.
6. Client public key: The public key to use for signing the source-authenticated group communication. These public key may be optionally certified using the AS key or a domain root key. This reduces the need for additional per-device public key storage on the receivers.
7. Algorithm: Used for source-authenticated secure group communication.
8. Issued at: Indicates date and time when the AT-R was created by the authorization server.

#### 4.3. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

- \* Client ID (unencrypted, integrity protected): Every sender managed by the AS MUST have a unique client ID.
  - \* Sequence Number (unencrypted, integrity protected): Used for replay protection.
2. Signature (not integrity protected): For source-authenticated integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.
2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.
3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.
4. Key ID (unencrypted, integrity protected): Is used to select the correct security association containing the verification key from the sequence of security associations identified by the application group id.
5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)
6. CoAP Payload (encrypted, integrity protected).

#### 4.4. Receiver Algorithm

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association to use to verify the message.

The key id is used for situations where the client may have different keys for different applications.

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the

request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- o Application Group ID lookup does not return any security association.
- o Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- o Integrity check fails.
- o Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

#### 4.5. Sender Algorithm

Figure 6 describes the algorithm for obtaining the necessary credentials to transmit a source-authenticated secure group message. When the sender wants to send a message to the application group, it checks if it has the respective signing key that matches the KID in the AT-R. If no signing key is available then it contacts the authorization server to obtain the AT-R and corresponding signing keys. Note that this assumes that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported.

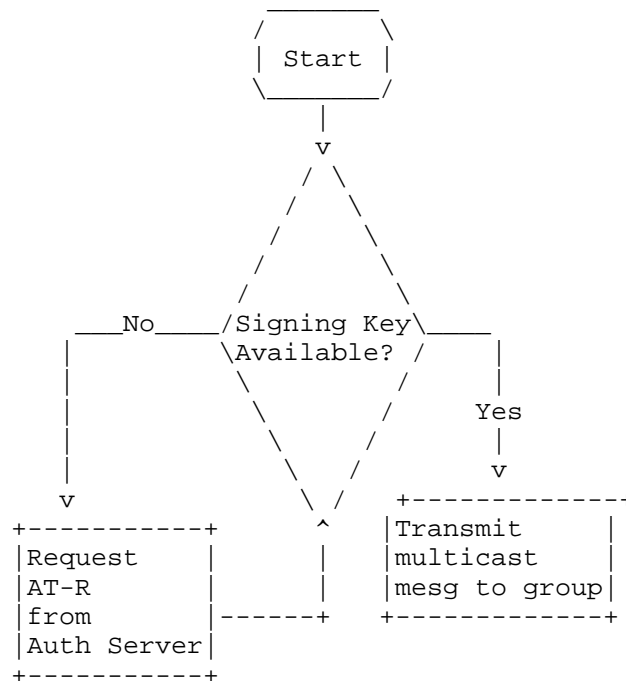


Figure 6: Steps to Transmit Source-authenticated Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a AT-R; the sender is likely to be pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

## 5. Security Considerations

### 5.1. Applicability statement

This document describes two architectures based on symmetric group keys in Section 3 and asymmetric keys in Section 4.

The symmetric key solution is based on a group key that is shared between all group members including senders and receivers. As all members of the group possess the same key, it is only possible to authenticate group membership for the source of a message. In particular, it is not possible to authenticate the unique source of a message and consequently it is not possible to authorize a single



node to control a group. Moreover, because the group key is shared across multiple nodes, it may be easier for an attacker to determine the group key by attacking any member of the group (note that this group key is dynamically generated and is usually stored in volatile memory which offers some additional protection). Subsequent to such an attack, it is also difficult to determine which of the group members was compromised and this makes it difficult to return the system to normal operation after an attack.

The asymmetric key solution distinguishes between a sender in the group and the receivers. In particular, the sender is in possession of a private key and the receivers are in possession of the corresponding public key. This allows the unique source of any group message to be authenticated. Moreover, an attacker cannot compromise the system by breaking into any of the receiving nodes. However, for constrained devices, the asymmetric key solution comes at a processing cost with cryptographic computations taking too long.

Therefore, it is recommended that whenever possible, the architecture with source authentication SHOULD be used to secure all multicast communication. However, in less sensitive applications (e.g. controlling luminaires in non-emergency applications), the architecture without source authentication MAY be used. When using the symmetric key solution two mitigating factors could improve system security. It is possible to achieve source authentication of messages at lower layers by requiring unique MAC layer keys for all devices within the network. The symmetric group keys are dynamically generated and therefore SHOULD be stored in volatile memory.

## 5.2. Token Verification

Due to the low latency requirements, token verification needs to be done locally and cannot be outsourced to other parties. For this reason a self-contained token must be used and the receivers are required to follow the steps outlined in Section 7.2 of RFC 7519 [RFC7519]. This includes the verification of the message authentication code protecting the contents of the token and the encryption envelope protecting the contained symmetric group key.

## 5.3. Token Revocation

Tokens have a specific lifetime. Setting the lifetime is a policy decision that involves making a trade-off decision. Allowing a longer lifetime increases the need to introduce a mechanism for token revocation (e.g., a real-time signal from the KDC/Authorization Server to the receivers to blacklist tokens) but lowers the communication overhead during normal operation since new tokens need to be obtained only from time to time. Real-time communication with

the receivers to revoke tokens may not be possible in all cases either, particularly when off-line operation is demanded or in small networks where the AS or even the KDC is only present during commissioning time.

We therefore recommend to issue short-lived tokens for dynamic scenarios like users accessing the lighting infrastructure of buildings using smartphones, tablets and alike to avoid potential security problems when tokens are leaked or where authorization rights are revoked. For senders that are statically mounted (like traditional light switches) we recommend a longer lifetime since re-configurations and token leakage is less likely to happen frequently.

To limit the authorization rights, tokens should contain an audience restriction, scoping their use to the intended receivers and to their access level.

#### 5.4. Time

Senders and receivers are not assumed to be equipped with real-time clocks but these devices are still assumed to interact with a time server. The lack of accurate clocks is likely to lead to clock drifts and limited ability to check for replays. For those cases where no time server is available, such as in small network installations, token verification cannot check for expired tokens and hence it might be necessary to fall-back to tokens that do not expire.

### 6. Operational Considerations

#### 6.1. Persistence of State Information

Devices in the lighting system can often be powered down intentionally or unintentionally. Therefore the devices may need to store the authorization tokens and cryptographic keys (along with replay context) in persistent storage like flash. This is especially required if the authorization server is no more online because it was removed after the commissioning phase. However the decision on the data to be persistently stored is a trade-off between how soon the devices can be back online to normal operational mode and the memory wear caused due to limited program-erase cycles of flash over the 15-20 years life-time of the device.

The different data that may need to be stored are access tokens AT-KDC, AT-R and last seen replay counter.

## 6.2. Provisioning in Small Networks

In small networks the authorization server and the KDC may be available only temporarily during the commissioning process and are not available afterwards.

## 6.3. Client IDs

A single device should not be managed by multiple KDCs. However, a group of devices in a domain (such as a lighting installation within an enterprise) should either be managed by a single KDC or, if there are multiple KDCs serving the devices in a given domain, these KDCs MUST exchange information so that the assigned client id and application group id values are unique within the devices in that domain. We assume that only devices within a given domain communicate with each other using group messages.

## 6.4. Application Groups vs. Security Groups

Multiple application groups may use the same key for performance reasons, reducing the number of keys needed to be stored - leading to less RAM needed by each node. This is only a reasonable option if the attack surface is not increased. For example, a room A is configured to use three application groups to address a subset of the device. In addition to configuring all nodes in room A with these three application groups the nodes are configured with a special group that allows them to access all devices in room A, referred as the all-nodes-in-room-A group. In this case, having the nodes to use the same key for the all-nodes-in-room group and the three groups does not increase the attack surface since any node can already use the all-nodes-in-room-A group to control other devices in that room. The three application groups in room A are a subset of the larger all-nodes-in-room-A group.

## 6.5. Lost/Stolen Device

The following procedure MUST be implemented if a device is stolen or keys are lost.

1. The AS tells the KDC to invalidate the AT-KDC.
2. The KDC no longer returns a new group key if the invalidated AT-KDC is presented to it.
3. The KDC generates new keys for all security groups to which the compromised device belongs.

The KDC SHOULD inform all devices in the security group to update their group key. This requires the KDC to maintain a list of all devices that belong to the security group and to be able to contact them reliably.

## 7. Acknowledgements

The author would like to thank Esko Dijk for his help with this document.

Parts of this document are a byproduct of the OpenAIS project, partially funded by the Horizon 2020 programme of the European Commission. It is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of fitness for a particular purpose. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the OpenAIS project or the European Commission.

## 8. IANA Considerations

This document defines one CoAP Header Option Application Group ID that MUST be allocated in the Registry "CoAP Option Numbers" of [RFC6749]. IANA is requested to allocation TBD option number to application group ID in this specification.

## 9. References

### 9.1. Normative References

[I-D.ietf-ace-actors]

Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-04 (work in progress), September 2016.

[I-D.ietf-cose-msg]

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

[I-D.wahlstroem-ace-cbor-web-token]

Wahlstroem, E., Jones, M., and H. Tschofenig, "CBOR Web Token (CWT)", draft-wahlstroem-ace-cbor-web-token-00 (work in progress), December 2015.

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

## 9.2. Informative References

- [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-08 (work in progress), July 2016.
- [I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-06 (work in progress), October 2016.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<http://www.rfc-editor.org/info/rfc6750>>.
- [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. Access Levels

A characteristic of the lighting domain is that access control decisions are also impacted by the type of operation being performed and those categories are listed below. The following access levels are pre-defined.

### Level 0: Service detection only

This is a service that is used with broadcast service detection methods. No operational data is accessible at this level.

### Level 1: Reporting only

This level allows access to sensor and other (relatively uncritical) operational data and the device error status. The operation of the system cannot be influenced using this level.

#### Level 2: Standard use

This level allows access to all operational features, including access to operational parameters. This is the highest level of access that can be obtained using (secure) multicast.

#### Level 3: Commissioning use / Parametrization Services

This level gives access to certain parameters that change the day-to-day operation of the system, but does not allow structural changes.

#### Level 4: Commissioning use / Localization and Addressing Services

(including Factory Reset) This level allows access to all services and parameters including structural settings.

#### Level 5: Software Update and related Services

This level allows the change and upgrade of the software of the devices.

Note: The use of group security is disallowed for level higher than Level 2 and unicast communication is used instead.

#### Authors' Addresses

Abhinav Somaraju  
Tridonic GmbH & Co KG  
Farbergasse 15  
Dornbirn 6850  
Austria

Email: [abhinav.somaraju@tridonic.com](mailto:abhinav.somaraju@tridonic.com)

Sandeep S. Kumar  
Philips Research  
High Tech Campus 34  
Eindhoven 5656 AE  
Netherland

Email: [ietf.author@sandeep-kumar.org](mailto:ietf.author@sandeep-kumar.org)

Hannes Tschofenig  
ARM Ltd.  
Hall in Tirol 6060  
Austria

Email: [Hannes.tschofenig@gmx.net](mailto:Hannes.tschofenig@gmx.net)  
URI: <http://www.tschofenig.priv.at>

Walter Werner  
Werner Management Services e.U.  
Josef-Anton-Herrburgerstr. 10  
Dornbirn 6850  
Austria

Email: [werner@werner-ms.at](mailto:werner@werner-ms.at)

ACE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: June 6, 2016

E. Wahlstroem  
Nexus Technology  
M. Jones  
Microsoft  
H. Tschofenig  
ARM Ltd.  
December 4, 2015

CBOR Web Token (CWT)  
draft-wahlstroem-ace-cbor-web-token-00

Abstract

CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. CWT is a profile of the JSON Web Token (JWT) that is optimized for constrained devices. The claims in a CWT are encoded in the Concise Binary Object Representation (CBOR) and CBOR Object Signing and Encryption (COSE) is used for added application layer security protection. A claim is a piece of information asserted about a subject and is represented as a name/value pair consisting of a claim name and a claim value.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 6, 2016.

Copyright Notice

Copyright (c) 2015 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 (<http://trustee.ietf.org/license-info>) in effect on the date of



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

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Claims . . . . .	3
3.1. Claim Names . . . . .	4
3.1.1. iss (Issuer) Claim . . . . .	4
3.1.2. sub (Subject) Claim . . . . .	4
3.1.3. aud (Audience) Claim . . . . .	4
3.1.4. exp (Expiration Time) Claim . . . . .	4
3.1.5. nbf (Not Before) Claim . . . . .	4
3.1.6. iat (Issued At) Claim . . . . .	5
3.1.7. cti (CWT ID) Claim . . . . .	5
4. Summary of the values, CBOR major types and encoded claim keys . . . . .	5
5. Security Considerations . . . . .	5
6. IANA Considerations . . . . .	6
7. Normative References . . . . .	6
Appendix A. Examples . . . . .	6
A.1. CWT with "aud" and symmetric key . . . . .	7
A.2. CWT with "aud" and EC key . . . . .	8
A.3. Full CWT . . . . .	10
Appendix B. Acknowledgements . . . . .	12
Appendix C. Document History . . . . .	12
Authors' Addresses . . . . .	13

## 1. Introduction

The JSON Web Token (JWT) [5] is a standardized security token format that has found use in OAuth 2.0 and OpenID Connect deployments, among other applications. JWT uses JSON Web Signatures (JWS) [3] and JSON Web Encryption (JWE) [4] to secure the contents of the JWT, which is a set of claims represented in JSON [5]. The use of JSON for encoding information is popular for Web and native applications, but it is considered inefficient for some Internet of Things (IoT) systems that use low power radio technologies.

In this document an alternative encoding of claims is defined. Instead of using JSON, as provided by JWTs, this specification uses CBOR [6] and calls this new structure "CBOR Web Token (CWT)", which is a compact means of representing secured claims to be transferred

between two parties. CWT is closely related to JWT. It references the JWT claims and both its name and pronunciation are derived from JWT. To protect the claims contained in CWTs, the CBOR Object Signing and Encryption (COSE) [7] specification is used.

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

## 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 "Key words for use in RFCs to Indicate Requirement Levels" [1].

This document reuses terminology from JWT [5] and COSE [7].

Type3StringOrURI:

The "Type3StringOrURI" term has the same meaning, syntax, and processing rules as the "StringOrUri" term defined in Section 2 of JWT [5], except that Type3StringOrURI uses CBOR major type 3 instead of a JSON string value.

FIXME: Use tag 32 for URIs?

Type6NumericDate:

The "Type6NumericDate" term has the same meaning, syntax, and processing rules as the "NumericDate" term defined in Section 2 of JWT [5], except that Type6NumericDate uses CBOR major type 6, with tag value 1, instead of a numeric JSON value.

CBOR encoded claim key:

The key used to identify a claim value.

## 3. Claims

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.

To keep CWTs as small as possible, the CBOR encoded claim keys are represented using CBOR major type 0. Section 4 summarizes all keys used to identify the claims defined in this document.

### 3.1. Claim Names

None of the claims defined below are intended to be mandatory to use or implement. They rather 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.

#### 3.1.1. iss (Issuer) Claim

The "iss" (issuer) claim has the same meaning, syntax, and processing rules as the "iss" claim defined in Section 4.1.1 of JWT [5], except that the format MUST be a Type3StringOrURI. The CBOR encoded claim key 1 MUST be used to identify this claim.

#### 3.1.2. sub (Subject) Claim

The "sub" (subject) claim has the same meaning, syntax, and processing rules as the "sub" claim defined in Section 4.1.2 of JWT [5], except that the format MUST be a Type3StringOrURI. The CBOR encoded claim key 2 MUST be used to identify this claim.

#### 3.1.3. aud (Audience) Claim

The "aud" (audience) claim has the same meaning, syntax, and processing rules as the "aud" claim defined in Section 4.1.3 of JWT [5], except that the format MUST be a Type3StringOrURI. The CBOR encoded claim key 3 MUST be used to identify this claim.

#### 3.1.4. exp (Expiration Time) Claim

The "exp" (expiration time) claim has the same meaning, syntax, and processing rules as the "exp" claim defined in Section 4.1.4 of JWT [5], except that the format MUST be a Type6NumericDate. The CBOR encoded claim key 4 MUST be used to identify this claim.

#### 3.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim has the same meaning, syntax, and processing rules as the "nbf" claim defined in Section 4.1.5 of JWT [5], except that the format MUST be a Type6NumericDate. The CBOR encoded claim key 5 MUST be used to identify this claim.

### 3.1.6. iat (Issued At) Claim

The "iat" (issued at) claim has the same meaning, syntax, and processing rules as the "iat" claim defined in Section 4.1.6 of JWT [5], except that the format MUST be a Type6NumericDate. The CBOR encoded claim key 6 MUST be used to identify this claim.

### 3.1.7. cti (CWT ID) Claim

The "cti" (CWT ID) claim has the same meaning, syntax, and processing rules as the "jti" claim defined in Section 4.1.7 of JWT [5], except that the format MUST be of major type 3 with a case-sensitive string value. The CBOR encoded claim key 7 MUST be used to identify this claim.

## 4. Summary of the values, CBOR major types and encoded claim keys

Claim	CBOR encoded claim key	CBOR major type of value
iss	1	3
sub	2	3
aud	3	3
exp	4	6 tag value 1
nbf	5	6 tag value 1
iat	6	6 tag value 1
cti	7	3

Figure 1: Summary of the values, CBOR major types and encoded claim keys.

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

## 5. Security Considerations

The security of the CWT is dependent on the protection offered by COSE. Without protecting the claims contained in a CWT an adversary is able to modify, add or remove claims. Since the claims conveyed in a CWT are used to make authorization decisions it is not only important to protect the CWT in transit but also to ensure that the recipient is able to authenticate the party that collected the claims and created the CWT. Without trust of the recipient in the party that created the CWT no sensible authorization decision can be made. Furthermore, the creator of the CWT needs to carefully evaluate each claim value prior to including it in the CWT so that the recipient can be assured about the correctness of the provided information.

## 6. IANA Considerations

This section will create a registry for CWT claims, possibly relating them to the JWT Claims Registry.

## 7. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [2] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [3] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<http://www.rfc-editor.org/info/rfc7515>>.
- [4] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.
- [5] 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>>.
- [6] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [7] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-08 (work in progress), November 2015.
- [8] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0", draft-seitz-ace-oauth-authz-00 (work in progress), October 2015.

## Appendix A. Examples

Three examples of CWTs follow.

## A.1. CWT with "aud" and symmetric key

A CWT used in the context of ACE requires at least the "aud" and a "cks" claim (defined elsewhere). This means that "iss", "alg", "key\_ops" and others are pre-established and assumed. This would look like this non-normative JSON.

```
{
  "aud": "coap://light.example.com",
  "cks": [
    // COSE_Key is a CBOR map with an array of keys
    {
      "kty": 4,           // symmetric key is indicated using kty 4
      "k": "loremipsum"  // the symmetric key
    }
  ]
}
```

Figure 2: "aud" claim and symmetric key in non-normative JSON

Using the CBOR encoded claim keys according to Section 4 and COSE [7] makes a CWT with "aud" and a symmetric key look like this in CBOR diagnostic notation:

```
{
  3: "coap://light.example.com",
  8: [
    {
      1: 4,
      -1: "loremipsum"
    }
  ]
}
```

Figure 3: CWT in CBOR diagnostic notation

Defined in CBOR.

```

a2                                # map(2)
  03                              # unsigned(3)
  78 18                          # text(24)
    636f61703a2f2f6c696768742e6578616d706c652e636f6d # "coap://light.example.c
om"
  08                              # unsigned(8)
  81                              # array(1)
    a2                            # map(2)
      01                          # unsigned(1)
      04                          # unsigned(4)
      20                          # negative(0)
      6a                          # text(10)
        6c6f72656d697073756d    # "loremipsum"

```

Figure 4: CWT with "aud" and symmetric key in CBOR

Size of the CWT with a symmetric key of 10 bytes is 45 bytes. This is then packaged signed and encrypted using COSE.

#### A.2. CWT with "aud" and EC key

Token with "aud" set to "coap://light.example.com" and an EC key with "kid" set to "11".

```

{
  "aud": "coap://light.example.com",
  "cks": [
    // COSE_Key is a CBOR map with an array of keys
    {
      "kty": "EC",
      "kid": "11",
      "crv": 1, // using P-384
      "x": h'bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff',
      "y": h'20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e'
    }
  ]
}

```

Figure 5: "aud" claim and EC key in non-normative JSON

Using the CBOR encoded claim keys according to Section 4 and COSE [7] makes a CWT with "aud" and an EC key look like this in CBOR diagnostic notation:

```
{
  3: "coap://light.example.com",
  8:
  [
    {
      1: 2,
      2: "11",
      -1: 1,
      -2: h'bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff',
      -3: h'20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e'
    }
  ]
}
```

Figure 6: CWT with EC key in CBOR diagnostic notation

Defined in CBOR.

```
a2                                # map(2)
  03                                # unsigned(3)
  78 18                            # text(24)
    636f61703a2f2f6c696768742e6578616d706c652e636f6d # "coap://light.example.c
om"
  08                                # unsigned(8)
  81                                # array(1)
    a5                              # map(5)
      01                            # unsigned(1)
      02                            # unsigned(2)
      02                            # unsigned(2)
      62                            # text(2)
        3131                        # "11"
      20                            # negative(0)
      01                            # unsigned(1)
      21                            # negative(1)
      58 20                         # bytes(32)
        bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff # "
\xBA\xC5\xB1\x1C\xAD\x8F\x99\xF9\xC7+\x05\xCFK\x9E&\xD2D\xDC\x18\x9FtR(%Z!\x9A\x
86\xD6\xA0\x9E\xFF"
      22                            # negative(2)
      58 20                         # bytes(32)
        20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e # "
\x13\x8B\xF8-\xC1\xB6\xD5b\xBE\x0F\xA5J\xB7\x80J:d\xB6\xD7,\xCF\xEDko\xB6\xED(\x
BB\xFC\x11~"
```

Figure 7: CWT with EC in CBOR

Size of the CWT with an EC key is 109 bytes. This is then packaged signed and encrypted using COSE.



## A.3. Full CWT

CWT using all claims defined by this specification, plus extensions for AIF and an EC key.

```
{
  "iss": "coap://as.example.com",
  "aud": "coap://light.example.com",
  "sub": "erikw",
  "exp": 1444064944,
  "nbf": 1443944944,
  "iat": 1443944944,
  "cti": 2929,
  "cks":
    [
      // COSE_Key is a CBOR map with an array of keys
      {
        "kty": "EC",
        "kid": "11",
        "crv": 1, // using P-384
        "x": h'bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff',
        "y": h'20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e',
      }
    ],
  "aif": [ ["/s/light", 1], ["/a/led", 5], ["/dtls", 2] ]
}
```

Figure 8: All claims, "aif" and EC key in non-normative JSON

Using the CBOR encoded claim keys according to Section 4 and COSE [7] makes a full CWT look like this in CBOR diagnostic notation:

```

{
  1: "coap://as.example.com",
  3: "coap://light.example.com",
  2: "erikw",
  4: 1(1444064944),
  5: 1(1443944944),
  6: 1(1443944944),
  7: 2929,
  8: [
    {
      1: 2,
      2: "11",
      -1: 1,
      -2: h'bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff',
      -3: h'20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e'
    }
  ],
  9: [ ["/s/light", 1], ["/a/led", 5], ["/dtls", 2] ]
}

```

Figure 9: Full CWT with EC key in CBOR diagnostic notation

Defined in CBOR.

```

a9                                # map(9)
01                                # unsigned(1)
75                                # text(21)
    636f61703a2f2f61732e6578616d706c652e636f6d # "coap://as.example.com"
03                                # unsigned(3)
78 18                             # text(24)
    636f61703a2f2f6c696768742e6578616d706c652e636f6d # "coap://light.example.c
om"
02                                # unsigned(2)
65                                # text(5)
    6572696b77                    # "erikw"
04                                # unsigned(4)
c1                                # tag(1)
    1a 5612aeb0                    # unsigned(1444064944)
05                                # unsigned(5)
c1                                # tag(1)
    1a 5610d9f0                    # unsigned(1443944944)
06                                # unsigned(6)
c1                                # tag(1)
    1a 5610d9f0                    # unsigned(1443944944)
07                                # unsigned(7)
19 0b71                           # unsigned(2929)
08                                # unsigned(8)

```

```

81                                # array(1)
  a5                             # map(5)
    01                           # unsigned(1)
    02                           # unsigned(2)
    02                           # unsigned(2)
    62                           # text(2)
      3131                       # "11"
    20                           # negative(0)
    01                           # unsigned(1)
    21                           # negative(1)
    58 20                       # bytes(32)
      bac5b11cad8f99f9c72b05cf4b9e26d244dc189f745228255a219a86d6a09eff # "
\xBA\xC5\xB1\x1C\xAD\x8F\x99\xF9\xC7+\x05\xCFK\x9E&\xD2D\xDC\x18\x9FtR(%Z!\x9A\x
86\xD6\xa0\x9E\xFF"
    22                           # negative(2)
    58 20                       # bytes(32)
      20138bf82dc1b6d562be0fa54ab7804a3a64b6d72ccfed6b6fb6ed28bbfc117e # "
\x13\x8B\xF8-\xC1\xB6\xD5b\xBE\x0F\xA5J\xB7\x80J:d\xB6\xD7,\xCF\xEDko\xB6\xED(\x
BB\xFC\x11~"
    09                           # unsigned(9)
    83                           # array(3)
      82                         # array(2)
        68                     # text(8)
          2f732f6c69676874     # "/s/light"
        01                     # unsigned(1)
      82                         # array(2)
        66                     # text(6)
          2f612f6c6564         # "/a/led"
        05                     # unsigned(5)
      82                         # array(2)
        65                     # text(5)
          2f64746c73           # "/dtls"
        02                     # unsigned(2)

```

Figure 10: Full CWT with EC in CBOR

Size of the CWT with an EC key is 194 bytes. This is then packaged signed and encrypted using COSE.

## Appendix B. Acknowledgements

A straw man proposal of CWT was written in the draft "Authorization for the Internet of Things using OAuth 2.0" [8] with the help of Ludwig Seitz, Goeran Selander, and Samuel Erdtman.

## Appendix C. Document History

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

-00

- o Created the initial version based on draft-wahlstroem-oauth-cbor-web-token-00.
- o Now reference the JWT claim definitions, rather than repeating them.

#### Authors' Addresses

Erik Wahlstroem  
Nexus Technology  
Sweden

Email: [erik.wahlstrom@nexusgroup.com](mailto:erik.wahlstrom@nexusgroup.com)  
URI: <https://www.nexusgroup.com>

Michael B. Jones  
Microsoft

Email: [mbj@microsoft.com](mailto:mbj@microsoft.com)  
URI: <http://self-issued.info/>

Hannes Tschofenig  
ARM Ltd.  
Hall in Tirol 6060  
Austria

Email: [Hannes.Tschofenig@arm.com](mailto:Hannes.Tschofenig@arm.com)