Abstract

Constrained nodes are small devices which are limited in terms of processing power, memory, non-volatile storage and transmission capacity. Due to these constraints, commonly used security protocols are not easily applicable. Nevertheless, an authentication and authorization solution is needed to ensure the security of these devices.

Due to the limitations of the constrained nodes it is especially important to develop a light-weight security solution which is adjusted to the relevant security objectives of each participating party in this environment. Necessary security measures must be identified and applied where needed.

This document gives an overview of the necessary terminology and introduces the actors in an architecture as guidance for the development of authentication and authorization solutions for constrained environments. The actors represent the relationships between the logical functional entities involved.

We also present a problem description for authentication and authorization in constrained-node networks, i.e. networks where some devices have severe constraints on memory, processing, power and communication bandwidth.

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 September 26, 2015.

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 ............................................. 3
   1.1.  Terminology ........................................ 4
2.  Problem Statement ....................................... 4
3.  Security Objectives .................................... 5
4.  Authentication and Authorization .......................... 6
5.  Autonomous Communication .................................. 7
6.  Actors .................................................... 8
   6.1.  Constrained Level Actors ............................ 8
   6.2.  Principal Level Actors ............................... 9
   6.3.  Less-Constrained Level Actors ....................... 10
7.  Architecture Variants .................................... 11
8.  Kinds of Protocols ....................................... 14
   8.1.  Constrained Level Protocols ......................... 14
   8.2.  Less-Constrained Level Protocols ................... 15
9.  Introduction to Problem Description ........................ 15
   9.1.  Terminology ....................................... 15
10.  Background .............................................. 16
11.  Problem Description ..................................... 18
   11.1.  Authorization .................................... 19
   11.2.  Authentication ..................................... 19
1. Introduction

Constrained nodes are small devices with limited abilities which in many cases are made to fulfill a single simple task. They 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.

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.

The limitations of the constrained nodes ask for security mechanisms which take the special characteristics of constrained environments into account. Therefore, it is crucial to identify the tasks which must be performed to meet the security requirements in constrained scenarios. Moreover, these tasks need to be assigned to logical functional entities which perform the tasks: the actors in the architecture. Thus, relations between the actors and requirements for protocols can be identified.

In this document, an architecture is developed to represent the relationships between the logical functional entities involved.
1.1. Terminology

Readers are required to be familiar with the terms and concepts defined in [RFC4949].

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.

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

Actor: A logical functional entity that performs one or more tasks. Depending on the tasks an actor must perform, the device that contains the actor may need to have certain system resources available. Multiple actors may share, i.e. be present within, a device or even a piece of software.

Resource Server (RS): An entity which hosts and represents a Resource.

Client (C): An entity which attempts to access a resource on a Server.

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

Requesting Party (RqP): The principal that is in charge of the Client and controls permissions concerning authorized representations of a Resource.

Principal: An individual that is either RqP or RO or both.

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

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

Attribute Binding Authority: An entity that is authorized to validate claims about an entity.

2. Problem Statement

The scenario this document addresses can be summarized as follows:

- C wants to access R on a RS.
A priori, C and RS do not necessarily know each other and have no security relationship.

C and / or RS are constrained.

```
|-------|  -- requests resource ---> | RS  |
|-------|  <-- provides resource---  -------|
```

Figure 1: Basic Scenario

The security requirements of any specific version of this scenario will include one or more of:

- Rq0.1: No unauthorized entity has access to (or otherwise gains knowledge of) R.
- Rq0.2: C is exchanging status updates of a resource only with authorized resources. (When C attempts to access R, that access reaches an authorized R).

Rq0.1 requires authorization on the server side while Rq0.2 requires authorization on the client side.

3. Security Objectives

The security objectives that can be addressed by an authorization solution are confidentiality and integrity. Additionally, allowing only selected entities limits the burden on system resources, thus helping to achieve availability. Misconfigured or wrongly designed authorization solutions can result in availability breaches: Users might no longer be able to use data and services as they are supposed to.

Authentication mechanisms can achieve additional security objectives such as non-repudiation and accountability. They are not related to authorization and thus are not in scope of this draft, but still should be considered by Authenticated Authorization solutions. Non-repudiation and accountability may require authentication on 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 importance of a security objective depends on the application the authorization mechanism is used for. [I-D.ietf-ace-usecases] indicates that security objectives differ for the various constrained environment use cases.
In many cases, one participating party might have different security objectives than the other. However, to achieve a security objective, both parties must participate in providing a solution. E.g., if RqP requires the integrity of sensor value representations RS is hosting, both C and RS need to integrity-protect the transmitted data. Moreover, RS needs to protect the access to the sensor representation to prevent unauthorized users to manipulate the sensor values.

4. Authentication and Authorization

Authorization solutions aim at protecting the access to items of interest, e.g. hardware or software resources or data: They enable the principal of such a resource 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 use the term _authenticated authorization_ to refer to a synthesis of mechanism for authentication and authorization.

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

In scenarios where devices are communicating autonomously there is less need to uniquely identify an individual device. For a principal, the fact that a device belongs to a certain company or that it has a specific type (e.g. light bulb) is likely more important than that it has a unique identifier.

Principals (RqP and RO) need to decide about the required level of granularity for the authorization, ranging from _device authorization_ over _owner authorization_ to _binary authorization_ and _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 have the same access permissions.
Unrestricted authorization for an item of interest means that no
authorization mechanism is used (not even by authentication) and all
entities are able to access the item as they see fit. More fine-
grained authorization does not necessarily provide more security.
Principals need to consider that an entity should only be granted the
permissions it really needs to ensure the confidentiality and
integrity of resources.

For all cases where an authorization solution is needed (all but
Unrestricted Authorization), the authorizing party needs to be able
to authenticate the party that is to be authorized. Authentication
is therefore required for messages that contain representations of an
accessed item. More precisely, the authorizing party needs to make
sure that the receiver of a message containing a representation, and
the sender of a message containing a representation are authorized to
receive and send this message, respectively. To achieve this, the
integrity of these messages is required: Authenticity cannot be
assured if it is possible for an attacker to modify the message
during transmission.

In some cases, only one side (only the client side or only the server
side) requires the integrity and / or confidentiality of a resource
value. In these cases, principals may decide to use binary
authorization which can be achieved by an authentication mechanism or
even unrestricted authorization where no authentication mechanism is
required. However, as indicated in Section 3, the security
objectives of both sides must be considered. The security objectives
of one side can often only be achieved with the help of the other
side. E.g., if the server requires the confidentiality of a resource
representation, the client must make sure that it does not send
resource updates to parties other than the server. Therefore, the
client must at least use binary authorization.

5. Autonomous Communication

The use cases defined in [I-D.ietf-ace-usecases] demonstrate that
constrained devices are often used for scenarios where their
principals are not present at the time of the communication.
Moreover, these devices often do not have any user interfaces or
displays. Even if the principals are present at the time of access,
they may not be able to communicate directly with the device. The
devices therefore need to be able to communicate autonomously. In
some scenarios there is an active user at one endpoint of the
communication. Other scenarios ask for true machine to machine (M2M)
communication.

To achieve the principals’ security objectives, the devices must be
enabled to enforce the security policies of their principals.
6. Actors

This section describes the various actors in the architecture. 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: 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.

6.1. 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:

- Communicate in a secure way (provide for confidentiality and integrity of messages).
- Validate that an entity is an authorized source for R.
- Securely transmit an access request.

RS performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages).
- Validate the authorization of the requester to access the requested resource as requested.
- Securely transmit a response to an access request.

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 of 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.

-------- | C | -- requests resource ---> | RS | Constrained Level
-------- <-- provides resource--- --------

Figure 2: Constrained Level Actors

6.2. Principal Level Actors

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

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

RqP must fulfill the following task:

1. 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:

1. Configure for RS authorization information for accessing R.

-------- | RqP | | RO | Principal Level
-------- in charge of in charge of
-------- V V

-------- | C | -- requests resource ---> | RS | Constrained Level
-------- <-- provides resource--- --------

Figure 3: Constrained Level Actors and Principal Level Actors
6.3. 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 complexity level for actors is introduced. An actor on the less-constrained level belongs to the same security domain as its respective constrained level actor. They also have the same 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 source for R. CAS can do that because for C, CAS is the authority for claims about RS.

CAS performs the following tasks:

- Validate on the client side that an entity has certain attributes.
- Obtain authorization information about an entity from C’s principal (RqP) and provide it to C.
- 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:

- Validate on the server side that an entity has certain attributes.
- Obtain authorization information about an entity from RS’ principal (RO) and provide it to RS.
- Negotiate means for secure communication to communicate with RS.
Figure 4: Overview of all Complexity Levels

For more detailed graphics please consult the PDF version.

7. Architecture Variants

As mentioned in section Section 6, actors can 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:
If RS is located on a more powerful device, it can be combined with AS:

Figure 5: Combined C and CAS
Figure 6: Combined AS and RS

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

Figure 7: CAS combined with AS

8. 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.

8.1. 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 6.1). C and RS might not belong to the same security domain. Therefore, constrained level protocols need to work between different security domains.

FIXME

![Figure 8: Constrained Level Tasks](image)

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.

8.1.1. Cross Level Support Protocols

Protocols which operate between a constrained device on one side and the corresponding less constrained device on the other are considered to be (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.
8.2. 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) [RFC5246] can be used alongside or instead of CoAP and DTLS. Moreover, existing security solutions for authentication and authorization such as the Web Authorization Protocol (OAuth) [RFC6749] and Kerberos [RFC4120] can likely be used without modifications and there are no limitations for the use of a Public Key Infrastructure (PKI).

FIXME

Figure 9: Less-constrained Level Tasks

9. Introduction to Problem Description

Authorization is the process of deciding what an entity ought to be allowed to do. This memo is about properties of security protocols to enable explicit and dynamic authorization of clients to access a resource at a server, in particular in constrained environments when the client and/or server are constrained nodes.

Relevant use cases are provided in [I-D.ietf-ace-usecases], which also lists some authorization problems derived from the use cases. In this memo we present a more specific problem description for authentication and authorization in constrained RESTful environments together with a detailed set of assumptions and requirements (cf. Section 12).

9.1. Terminology

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

RESTful terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. are defined in [RFC7228].
"Explicit" authorization is used here to describe the ability to specify in some detail which entity has access to what and under what conditions, as opposed to "implicit" authorization where an entity is either allowed to access everything or nothing.

"Dynamic" authorization means that the access control policies and the parameters on which they are evaluated may change during normal operations, as opposed to "static" authorization meaning that access control policies cannot be changed during normal operations and may require some special procedure such as out-of-band provision.

10. Background

We assume a client-server setting, where a client wishes to access some resource hosted by a server. Such resources may e.g. be sensor data, configuration data, or actuator settings. Thus access to a resource could be by different methods, some of which change the state of the resource. In this memo, we consider the REST setting i.e. GET, POST, PUT and DELETE, and application protocols in scope are HTTP [RFC7231] and CoAP [RFC7252].

We assume that the roles of client and server are not fixed, i.e. a node which is client could very well be server in some other context and vice-versa. Further we assume that in some cases, clients are not previously known to servers, thus we cannot assume that the server has access control policies specific to that client when the client initiates communication.

Finally we also assume that in a significant number of cases, the server and/or the client are too constrained to handle the evaluation of complex access control policies and related configuration on their own. Many authorization solutions involve a centralized, trusted third party, supporting the client and/or resource server. A trusted third party provides a more scalable way to centrally manage authorization policies, in order to ensure consistent authorization decisions. The physical separation of policy decision and policy enforcement is an established principle in policy based management, e.g. [RFC2748].

Borrowing from OAuth 2.0 [RFC6749] terminology we name the entities: client (C), resource server (RS), authorization server (AS - the third party), and resource owner (RO). RO is in charge of the access control policies implemented in the AS governing the actions of RS. However, the RO need not be active in a constrained device access control setting, so we cannot rely on timely interactions with the RO. In the target setting RS is typically constrained, C may be constrained, whereas AS is not assumed to be constrained.
Since RS is constrained, we assume that it needs to offload authorization policy management and/or authorization decision making to AS. This means that some authorization information needs to be transferred from AS to RS.

Protecting information carried between AS and RS, requires some a priori established cryptographic keys. How those keys are established is out of scope for this problem description.

AS may for example be implemented as a cloud service, in a home server, or in a smartphone. C and RS may or may not have connectivity to AS at the time of the access request, e.g. because they cannot handle multiple, simultaneous connections. Another reason for intermittent connectivity may be that constant connectivity is not affordable (e.g. due to limited battery power, or a sensor mobility business case for which cellular connectivity cost too much or is not available). Obviously, in order for a client request to reach RS there must be connectivity between C and RS, but that could be a short range technology such as Bluetooth, ZigBee, or NFC. Furthermore, if there is not sufficient authorization information about C in RS, and neither C nor RS can access AS, access requests will be denied. Therefore we assume that either C or RS can access AS at some point in time, prior to the client’s request.

As a summary, there are potentially three information flows that needs to be protected (see Figure 10):

1. The transfer of authorization information from AS to RS
2. The transfer of cryptographic keys or credentials from AS to RS and C, respectively
3. The access request/response procedure between C and RS
The information flow in Figure 10 above enables RO to control the interactions of a constrained RS by means of access control policies. There is an ongoing discussion about an analogous information flow enabling the stakeholder associated to C ("Requesting Party" in UMA terminology [I-D.hardjono-oauth-umacore]) to control the interactions of a constrained C by means of policies. While this would not be policies for access control to resources, it could be useful in certain settings which require dynamically changing interaction patterns with a constrained client without updating firmware. Such a solution could potentially reuse all security components required to protect the information flow in 1., so no additional specifications would be needed. This aspect is not discussed further in this draft.

11. Problem Description

A number of problems needs to be solved in order to achieve explicit and dynamic authorization, as is described in this section.
11.1. Authorization

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

- AS needs to transfer authorization information to RS.
- 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.
- The RS needs to be able to verify the authenticity of the authorization information. There is a trade-off here between processing complexity and deployment complexity.
- The RS needs to enforce the authorization decisions of the AS. The authorization information it obtained from AS might require additional policy evaluation (e.g. matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the RS needs to be adapted to the capabilities of the constrained device.
- 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 RS at different times and in different ways prior to or during the client request.

11.2. Authentication

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

- RS need to authenticate AS to ensure that the authorization information and related data comes from the correct source.
- C may need to authenticate AS to ensure that it gets security information related to the resources from the right source.
- In some use cases RS needs to authenticate some property of C, in order to bind it to the relevant authorization information. In other use cases, authentication and authorization of C may be implicit, e.g. by encrypting the resource representation the RS only providing access to those who possess the key to decrypt.
o C may need to authenticate RS, in order to ensure that it is interacting with the right resources. Alternatively C may just verify the integrity of a received resource representation.

o AS may need to authenticate its communication partner (either C or RS), in order to ensure it serves the correct device.

11.3. 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, e.g. using [I-D.selander-ace-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.koster-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.

11.4. 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. Do we want to support solutions based on asymmetric keys or 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 e.g. in terms of deployment.

Key Establishment
How are the corresponding cryptographic keys established? Considering Section 11.1 there must be a binding 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 [I-D.ietf-ace-usecases] describes spontaneous change of access policies - e.g. 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?

12. 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.

12.1. Architecture

The architecture consists of at least the following types of nodes:

- RS hosting resources, and responding to access requests
- C requesting access to resources
- AS supporting the access request/response procedure by providing authorization information to RS.
  * AS may also provide other services such as authenticating C on behalf of RS, or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- The architecture may contain intermediary nodes between any pair of C, RS and AS, such as e.g. forward/reverse proxies in the CoRE architecture. The solution shall not unduly restrict the use of intermediaries.
  * The architecture shall support session based security and data object security.
12.2. Constrained Devices

- 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

- AS is not a constrained device.

- All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.
  - We assume the use of a standardized symmetric key algorithm, such as AES.
  - Except for the most constrained devices we assume the use of a standardized cryptographic hash function such as SHA-256.

- Public key cryptography requires additional resources (e.g. RAM, ROM, power, specialized hardware).

- A DTLS handshake involves significant computation, communication, and memory overheads in the context of constrained devices.
  - The RAM requirements of DTLS handshakes with public key cryptography are prohibitive for certain constrained devices.
  - Certificate-based DTLS handshakes require significant volumes of communication, RAM (message buffers) and computation.

- The solution shall support a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. section Section 13.2).
12.3. Authentication

- RS need to authenticate AS to ensure that the authorization information and related data comes from the correct source.
- Depending on use case, C, RS or AS may need to authenticate each other.

12.4. Authorization

- The authorization decision is based on credentials presented by C, the requested resource, the RESTful method, and local context in RS at the time of the request, or on any subset of this information.
- The authorization decision is taken either by AS or RS.
- The authorization decision is enforced by RS.
  * RS needs to have access to 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.
- Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (e.g. resource descriptions).
- The solution may need to be able to support the delegation of access rights.

12.5. Authorization Information

- Authorization information is transferred from AS to RS using Agent, Push or Pull mechanisms [RFC2904].
- RS shall authenticate that the authorization information is coming from AS.
- The authorization information may also be encrypted end-to-end between AS and RS.
- RS may not be able to communicate with AS at the time of the request from C.
- RS may store or cache authorization information.
o Authorization information may be pre-configured in RS.

o Authorization information stored or cached in RS shall be possible to change. The change of such information shall be subject to authorization.

o Authorization policies stored on RS may be handled as a resource, i.e. information located at a particular URI, accessed with RESTful methods, and the access being subject to the same authorization mechanics. AS may have special privileges when requesting access to the authorization policy resources on RS.

o There may be mechanisms for C to look up the AS which provides authorization information about a particular resource.

12.6. Resource Access

o Resources are accessed in a RESTful manner using GET, PUT, POST, DELETE.

o By default, the resource request shall be integrity protected and may be encrypted end-to-end from C to RS. It shall be possible for RS to detect a replayed request.

o By default, the response to a request shall be integrity protected and encrypted end-to-end from RS to C. It shall be possible for C to detect a replayed response.

o RS shall be able to verify that the request comes from an authorized client

o C shall 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).

12.7. Keys and Cipher Suites

o AS and RS have established cryptographic keys. Either AS and RS 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 can be protected with symmetric and/or asymmetric keys.
There must be a mechanism for RS to establish the necessary key(s) to verify and decrypt the request.

There must be a mechanism for C to establish the necessary key(s) to verify and decrypt the response.

There must be a mechanism for C to look up the supported cipher suites of a RS.

12.8. Network Considerations

- The solution shall prevent network overload due to avoidable communication with AS.
- The solution shall prevent network overload by compact authorization information representation.
- The solution shall optimize the case where authorization information does not change often.
- The solution where possible shall support an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

12.9. Legacy Considerations

- The solution shall work with existing infrastructure.
- The solution shall support authorization of access to legacy devices.

13. Security Considerations

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

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments are provided in e.g. OAuth 2.0 [RFC6749].

In this section we focus on specific security aspects related to authorization in constrained-node networks.
13.1. Physical Attacks on Sensor and Actuator Networks

The focus of this work is on constrained-node networks consisting of connected 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:

- Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.
- 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. For example:

- Instead of eavesdropping the sensor data or attacking the authorization system to gain access to the data, the attacker could make its own measurements on the physical object.
- Instead of manipulating the sensor data the attacker could change the physical object which the sensor is measuring, thereby changing the payload data which is being sent.
- Instead of manipulating data for an actuator or attacking the authorization system, the attacker could perform an unauthorized action directly on the physical object.
- A denial-of-service attack could be performed physically on the object or device.

All 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.)

As a conclusion, if an attacker has physical access to a sensor or actuator device, then much of the security functionality elaborated in this draft is not effective to protect the asset during the physical attack.
Since it does not make sense to design a solution for a situation that cannot be protected against we assume there is no need to protect assets which are exposed during a physical attack. In other words, either an attacker does not have physical access to the sensor or actuator device, or if it has, the attack shall only have effect during the period of physical attack.

13.2. Time Measurements

Measuring time with certain accuracy is important to achieve certain security properties, for example to determine whether a public key certificate, access token or some other assertion is valid.

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 can connect directly to AS it could get updated in terms of time as well as revocation information.

If RS continuously measures time but can’t connect to AS or other trusted source, time drift may have to be accepted and it may not be able to manage revocation. However, it 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 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.

14. IANA Considerations

This document has no actions for IANA.

15. Acknowledgements

The authors would like to thank Olaf Bergmann, Robert Cragie, Klaus Hartke, Sandeep Kumar, John Mattson, Corinna Schmitt, Mohit Sethi, 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 acknowledge input provided by Hummen et al. [HUM14delegation].
16. Informative References

[HUM14delegation]

[I-D.hardjono-oauth-umacore]

[I-D.ietf-ace-usecases]

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

[I-D.selander-ace-object-security]


Authors’ Addresses

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

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

Ludwig Seitz
SICS Swedish ICT AB
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@sics.se
Delegated CoAP Authentication and Authorization Framework (DCAF)
draft-gerdes-ace-dcaf-authorize-02

Abstract

This specification defines a protocol for delegating client
authentication and authorization in a constrained environment for
establishing a Datagram Transport Layer Security (DTLS) channel
between resource-constrained nodes. The protocol relies on DTLS to
transfer authorization information and shared secrets for symmetric
cryptography between entities in a constrained network. A resource-
constrained node can use this protocol to delegate authentication of
communication peers and management of authorization information to a
trusted host with less severe limitations regarding processing power
and memory.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at 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 September 10, 2015.

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 . . . . . . . . . . . . . . . . . . . . . . . . .  3
   1.1.  Features . . . . . . . . . . . . . . . . . . . . . . . . .  4
   1.2.  Terminology . . . . . . . . . . . . . . . . . . . . . . .  4
       1.2.1.  Actors . . . . . . . . . . . . . . . . . . . . . . .  4
       1.2.2.  Other Terms . . . . . . . . . . . . . . . . . . . .  5
2.  System Overview . . . . . . . . . . . . . . . . . . . . . . .  6
3.  Protocol . . . . . . . . . . . . . . . . . . . . . . . . . . .  6
   3.1.  Overview . . . . . . . . . . . . . . . . . . . . . . . . .  7
   3.2.  Unauthorized Resource Request Message . . . . . . . . .  8
   3.3.  SAM Information Message . . . . . . . . . . . . . . . .  9
   3.4.  Access Request . . . . . . . . . . . . . . . . . . . . . 10
   3.5.  Ticket Request Message . . . . . . . . . . . . . . . . . 11
   3.6.  Ticket Grant Message . . . . . . . . . . . . . . . . . . 12
   3.7.  Ticket Transfer Message . . . . . . . . . . . . . . . . . 13
   3.8.  DTLS Channel Setup Between C and S . . . . . . . . . . . 14
   3.9.  Authorized Resource Request Message . . . . . . . . . . 15
   3.10. Dynamic Update of Authorization Information . . . . . . . 16
       3.10.1. Handling of Ticket Transfer Messages . . . . . . . 17
4.  Ticket . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
   4.1.  Face . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
   4.2.  CI . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
   4.3.  Revocation . . . . . . . . . . . . . . . . . . . . . . . . 19
       4.3.1. Lifetime . . . . . . . . . . . . . . . . . . . . . . . 19
       4.3.2. Revocation Messages . . . . . . . . . . . . . . . . 20
5.  Payload Format and Encoding (application/dcaf+cbor) . . . . . 20
   5.1.  Examples . . . . . . . . . . . . . . . . . . . . . . . . . 23
6.  DTLS PSK Generation Methods . . . . . . . . . . . . . . . . . 25
   6.1.  DTLS PSK Transfer . . . . . . . . . . . . . . . . . . . . 25
   6.2.  Distributed Key Derivation . . . . . . . . . . . . . . . 25
7.  Authorization Configuration . . . . . . . . . . . . . . . . . . 26
8.  Trust Relationships . . . . . . . . . . . . . . . . . . . . . . 26
9.  Listing Authorization Manager Information in a Resource
    Directory . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
    9.1.  The "auth-request" Link Relation . . . . . . . . . . . . 27
10. Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
    10.1. Access Granted . . . . . . . . . . . . . . . . . . . . . 28
    10.2. Access Denied . . . . . . . . . . . . . . . . . . . . . . 30
    10.3. Access Restricted . . . . . . . . . . . . . . . . . . . . 31
    10.4. Implicit Authorization . . . . . . . . . . . . . . . . . 32
11. Specific Usage Scenarios .......................... 33
11.1. Combined Authorization Manager and Client ........ 33
11.1.1. Creating the Ticket Request Message ............ 33
11.1.2. Processing the Ticket Grant Message ............ 34
11.2. Combined Client Authorization Manager and Server Authorization Manager ....................... 34
11.2.1. Processing the Access Request Message .......... 35
11.2.2. Creating the Ticket Transfer Message .......... 35
11.3. Combined Server Authorization Manager and Server .... 35
12. Security Considerations ............................ 36
13. IANA Considerations ................................. 36
13.1. DTLS PSK Key Generation Methods .................. 37
13.2. dcaf+cbor Media Type Registration ............... 37
13.3. CoAP Content Format Registration ................ 38
14. References ............................................ 38
14.1. Normative References .............................. 39
14.2. Informative References ............................ 39
Authors’ Addresses ....................................... 40

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a transfer protocol similar to HTTP which is designed for the special requirements of constrained environments. A serious problem with constrained devices is the realization of secure communication. The devices only have limited system resources such as memory, stable storage (such as disk space) and transmission capacity and often lack input/output devices such as keyboards or displays. Therefore, they are not readily capable of using common protocols. Especially authentication mechanisms are difficult to realize, because the lack of stable storage severely limits the number of keys the system can store. Moreover, CoAP has no mechanism for authorization.

[I-D.gerdes-ace-actors] describes an architecture that is designed to help constrained nodes with authorization-related tasks by introducing less-constrained nodes. These Authorization Managers perform complex security tasks for their nodes such as managing keys for numerous devices, and enable the constrained nodes to enforce the authorization policies of their principals.

DCAF uses access tokens to implement this architecture. A device that wants to access an item of interest on a constrained node first has to gain permission in the form of a token from the node’s Authorization Manager.

As fine-grained authorization is not always needed on constrained devices, DCAF supports an implicit authorization mode where no authorization information is exchanged.
The main goals of DCAF are the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] between two nodes and to securely transmit authorization tickets.

1.1. Features

- Utilize DTLS communication with pre-shared keys.
- Authenticated exchange of authorization information.
- Simplified authentication on constrained nodes by handing the more sophisticated authentication over to less-constrained devices.
- Support of secure constrained device to constrained device communication.
- Authorization policies of the principals of both participating parties are ensured.
- Simplified authorization mechanism for cases where implicit authorization is sufficient.
- Using only symmetric encryption on constrained nodes.

1.2. 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 RFC 2119 [RFC2119].

Readers are expected to be familiar with the terms and concepts defined in [I-D.gerdes-ace-actors].

1.2.1. Actors

Server (S): An endpoint that hosts and represents a CoAP resource.

Client (C): An endpoint that attempts to access a CoAP resource on the Server.

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

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

Authorization Manager (AM): An entity that is either a SAM or a CAM.
Client Overseeing Principal (COP): The principal that is in charge of the Client and controls permissions concerning authorized representations of a CoAP resource.

Resource Overseeing Principal (ROP): The principal that is in charge of the CoAP resource and controls its access permissions.

1.2.2. Other Terms


Authorization information: Contains all information needed by S to decide if C is privileged to access a resource in a specific way.

Authentication information: Contains all information needed by S to decide if the entity in possession of a certain key is verified by SAM.

Access information: Contains authentication information and, if necessary, authorization information.

Access ticket: Contains the authentication and, if necessary, the authorization information needed to access a resource. A Ticket consists of the Ticket Face and the Client Information. The access ticket is a representation of the access information.

Ticket Face: The part of the ticket which is generated for the Server. It contains the authorization information and all information needed by the Server to verify that it was granted by SAM.

Client Information (CI): The part of the ticket which is generated for the Client. It contains the Verifier and optionally may contain authorization information that represent COP’s authorization policies for C.

Verifier: It enables the client to verify that it is communicating with an appropriate S.

Explicit authorization: SAM informs the S in detail which privileges are granted to the Client.

Implicit authorization: SAM authenticates the Client for the Server without specifying the privileges in detail. This can be used for binary or unrestricted authorization (cf section 4 of [I-D.gerdes-ace-actors]).
2. System Overview

Within the DCAF Architecture each Server (S) has a Server Authorization Manager (SAM) which conducts the authentication and authorization for S. S and SAM share a symmetric key which has to be exchanged initially to provide for a secure channel. The mechanism used for this is not in the scope of this document.

To gain access to a specific resource on a S, a Client (C) has to request an access ticket from the SAM serving S either directly or, if it is a constrained device, using its Client Authorization Manager (CAM). In the following, we always discuss the CAM role separately, even if that is co-located within a (more powerful) C (see section Section 11 for details about co-located actors).

CAM decides if S is an authorized source for R according to the policies set by COP and in this case transmits the request to SAM. If SAM decides that C is allowed to access the resource according to the policies set by ROP, it generates a DTLS pre-shared key (PSK) for the communication between C and S and wraps it into an access ticket. For explicit access control, SAM adds the detailed access permissions to the ticket in a way that CAM and S can interpret. CAM checks if the permissions in the access ticket comply with COP’s authorization policies for C, and if this is the case sends it to C. After C presented the ticket to S, C and S can communicate securely.

To be able to provide for the authentication and authorization services, an Authorization Manager has to fulfill several requirements:

- AM must have enough stable storage (such as disk space) to store the necessary number of credentials (matching the number of Clients and Servers).
- AM must possess means for user interaction, for example directly or indirectly connected input/output devices such as keyboard and display, to allow for configuration of authorization information by the respective Principal.
- AM must have enough processing power to handle the authorization requests for all constrained devices it is responsible for.

3. Protocol

The DCAF protocol comprises three parts:

1. transfer of authentication and, if necessary, authorization information between C and S;
2. transfer of access requests and the respective ticket grants between C and CAM; and

3. transfer of access requests and the respective ticket grants between SAM and CAM.

3.1. Overview

In Figure 1, a DCAF protocol flow is depicted (messages in square brackets are optional):

```
CAM                   C                    S                   SAM
| <== DTLS chan. ==> |                    | <== DTLS chan. ==> |
|                    | [Resource Req.-->] |                    |
|                    |                   | [<- SAM Info.]    |
|                    |                   |                    |
|                    | <- Access Req.   |                    |
|                    |                   |                    |
|                    | <== TLS/DTLS channel (CAM/SAM Mutual Authentication) ===> |
| Ticket Request     |                     | Ticket Grant       |
|                    |                     |                    |
| Ticket Transf. --> |                     |                    |
|                   |                     |                    |
|                   | <== DTLS chan. ==> | Auth. Res. Req. -> |
```

Figure 1: Protocol Overview

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

Instead of the initial Unauthorized Resource Request message, C MAY look up the desired resource in a resource directory (cf. [I-D.ietf-core-resource-directory]) that lists S’s resources as discussed in Section 9.

Once C knows SAM’s address, it can send a request for authorization to SAM using its own CAM. CAM and SAM authenticate each other and each determine if the request is to be authorized. If it is, SAM generates an access ticket for C. The ticket contains keying material for the establishment of a secure channel and, if necessary, a representation of the permissions C has for the resource. C keeps
one part of the access ticket and presents the other part to S to prove its right to access. With their respective parts of the ticket, C and S are able to establish a secure channel.

The following sections specify how CoAP is used to interchange access-related data between S and SAM so that SAM can provide C and S with sufficient information to establish a secure channel, and simultaneously convey authorization information specific for this communication relationship to S.

Note: Special implementation considerations apply when one single entity takes the role of more than one actors. Section 11 gives additional advice on some of these usage scenarios.

This document uses Concise Binary Object Representation (CBOR, [RFC7049]) to express authorization information as set of attributes passed in CoAP payloads. Notation and encoding options are discussed in Section 5.

3.2. Unauthorized Resource Request Message

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

- The request has been received on an insecure channel.
- S has no valid access ticket for the sender of the request regarding the requested action on that resource.
- S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access was granted and a secure channel has been established between C and S.

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Server MUST provide proper SAM Information to enable the Client to request an access ticket from S’s SAM as described in Section 3.3.

The response code MUST be 4.01 (Unauthorized) in case the sender of the Unauthorized Resource Request message is not authenticated, or if S has no valid access ticket for C. If S has an access ticket for C but not for the resource that C has requested, S MUST reject the
request with a 4.03 (Forbidden). If S has an access ticket for C but it does not cover the action C requested on the resource, S MUST reject the request with a 4.05 (Method Not Allowed).

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

3.3. SAM Information Message

The SAM Information Message is sent by S as a response to an Unauthorized Resource Request message (see Section 3.2) to point the sender of the Unauthorized Resource Request message to S’s SAM. The SAM information is a set of attributes containing an absolute URI (see Section 4.3 of [RFC3986]) that specifies the SAM in charge of S.

The message MAY also contain a timestamp generated by S.

Figure 2 shows an example for a SAM Information message payload using CBOR diagnostic notation. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://sam.example.com/authorize", TS: 168537}
```

Figure 2: SAM Information Payload Example

In this example, the attribute SAM points the receiver of this message to the URI "coaps://sam.example.com/authorize" to request access permissions. The originator of the SAM Information payload (i.e., S) uses a local clock that is loosely synchronized with a time scale common between S and SAM (e.g., wall clock time). Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention. Refer to Section 4.1 for more details concerning the usage of time stamps to ensure freshness of access tickets.

The examples in this document are written in CBOR diagnostic notation to improve readability. Figure 3 illustrates the binary encoding of the message payload shown in Figure 2.
3.4. Access Request

To retrieve an access ticket for the resource that C wants to access, C sends an Access Request to its CAM. The Access Request is constructed as follows:

1. The request method is POST.
2. The request URI is set as described below.
3. The message payload contains a data structure that describes the action and resource for which C requests an access ticket.

The request URI identifies a resource at CAM for handling authorization requests from C. The URI SHOULD be announced by CAM in its resource directory as described in Section 9.

Note: Where capacity limitations of C do not allow for resource directory lookups, the request URI in Access Requests could be hard-coded during provisioning or set in a specific device configuration profile.

The message payload is constructed from the SAM information that S has returned in its SAM Information message (see Section 3.3) and information that C provides to describe its intended request(s). The Access Request MUST contain the following attributes:

1. Contact information for the SAM to use.
2. An absolute URI of the resource that C wants to access.
3. The actions that C wants to perform on the resource.
4. Any time stamp generated by S.
An example Access Request from C to CAM is depicted in Figure 4. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
POST client-authorize
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://sam.example.com/authorize",
  SAI: ["coaps://temp451.example.com/s/tempC", 5],
  TS: 168537
}
```

Figure 4: Access Request Message Example

The example shows an Access Request message payload for the resource "/s/tempC" on the Server "temp451.example.com". Requested operations in attribute SAI are GET and PUT.

The attributes SAM (that denotes the Server Authorization Manager to use) and TS (a nonce generated by S) are taken from the SAM Information message from S.

The response to an Authorization Request is delivered by CAM back to C in a Ticket Transfer message.

### 3.5. Ticket Request Message

When CAM receives an Access Request message from C and COP specified authorization policies for C, CAM MUST check if the requested actions are allowed according to these policies. If this is not the case, CAM MUST send a 4.03 response.

If no authorization policies were specified or the requested action is allowed according to the authorization policies, CAM either returns a cached response or attempts to create a Ticket Request message.

CAM MAY return a cached response if it is known to be fresh according to Max-Age. CAM SHOULD NOT return a cached response if it expires in less than a minute.

If CAM does not send a cached response, it checks whether the request payload is of type "application/dcaf+cbor and contains at least the fields SAM and SAI. CAM MUST respond with 4.00 (Bad Request) if the type is "application/dcaf+cbor and any of these fields is missing or does not conform to the format described in Section 5. Content formats other than application/dcaf+cbor are out of scope of this specification.
If the payload is correct, CAM creates a Ticket Request message from the Access Request received from C as follows:

1. The destination of the Ticket Request message is derived from the authority information in the URI contained in field "SAM" of the Access Request message payload.

2. The request method is POST.

3. The request URI is constructed from the SAM field received in the Access Request message payload.

4. The payload is copied from the Access Request sent by C.

5. A label that describes the Client is added to the payload

To send the Ticket Request message to SAM a secure channel between CAM and SAM MUST be used. Depending on the URI scheme used in the SAM field of the Access Request message payload (the less-constrained devices CAM and SAM do not necessarily use CoAP to communicate with each other), this could be, e.g., a DTLS channel (for "coaps") or a TLS connection (for "https"). CAM and SAM MUST be able to mutually authenticate each other, e.g. based on a public key infrastructure. (Refer to Section 8 for a detailed discussion of the trust relationship between Client Authorization Managers and Server Authorization Managers.)

3.6. Ticket Grant Message

When SAM has received a Ticket Request message it has to evaluate the access request information contained therein. First, it checks whether the request payload is of type "application/dcaf+cbor" and contains at least the fields SAM and SAI. SAM MUST respond with 4.00 (Bad Request) for CoAP (or 400 for HTTP) if the type is "application/dcaf+cbor" and any of these fields is missing or does not conform to the format described in Section 5.

SAM decides whether or not access is granted to the requested resource and then creates a Ticket Grant message that reflects the result. To grant access to the requested resource, SAM creates an access ticket comprised of a Face and the Client Information as described in Section 4.1.

The Ticket Grant message then is constructed as a success response indicating attached content, i.e. 2.05 for CoAP, or 200 for HTTP, respectively. The payload of the Ticket Grant message is a data structure that contains the result of the access request. When access is granted, the data structure contains the Ticket Face, the
Client Information, which at this point only consists of the Verifier and the Session Key Generation Method.

The Ticket Grant message MAY provide cache-control options to enable intermediaries to cache the response. The message MAY be cached according to the rules defined in [RFC7252] to facilitate ticket retrieval when C has crashed and wants to recover the DTLS session with S.

SAM SHOULD set Max-Age according to the ticket lifetime in its response (Ticket Grant Message).

Figure 5 shows an example Ticket Grant message using CoAP. The Face/Verifier information is transferred as a CBOR data structure as specified in Section 5. The Max-Age option tells the receiving CAM how long this ticket will be valid.

```plaintext
2.05 Content
Content-Format: application/dcaf+cbor
Max-Age: 86400
{ 
  F: { 
    SAI: [ "/s/tempC", 7 ],
    TS: 0("2013-07-10T04:12.391"),
    L:  86400,
    G: hmac_sha256
  },
  V: h'f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9'
}
```

Figure 5: Example Ticket Grant Message

A Ticket Grant message that declines any operation on the requested resource is illustrated in Figure 6. As no ticket needs to be issued, an empty payload is included with the response.

```plaintext
2.05 Content
Content-Format: application/dcaf+cbor

Figure 6: Example Ticket Grant Message With Reject
```

3.7. Ticket Transfer Message

A Ticket Transfer message delivers the access information sent by SAM in a Ticket Grant message to the requesting client C. The Ticket Transfer message is the response to the Access Request message sent from C to CAM and includes any access information from SAM contained in the Ticket Grant message.
The Authorization Information provided by SAM in the Ticket Grant Message may grant more permissions than C has requested. The authorization policies of COP and ROP may differ: COP might want restrict the resources C is allowed to access, and the actions that C is allowed to perform on the resource.

If CAM must ensure authorization policies COP configured, CAM MUST add Authorization Information for C (CAI) to the CI. Since C and CAM use a DTLS channel for communication, the authorization information does not need to be encrypted.

CAM includes the Face and Verifier sent by SAM in the Ticket Transfer message. CAM MUST NOT include any other information SAM provided. In particular, CAM MUST NOT include any CAI information provided by SAM.

Figure 7 shows an example Ticket Transfer message that conveys the permissions for actions GET, POST, PUT (but not DELETE) on the resource "/s/tempC" in field SAI. As CAM only wants to permit outbound GET requests, it restricts C's permissions in the field CAI accordingly.

2.05 Content
Content-Format: application/dcaf+cbor
Max-Age: 86400

{ F: {    SAI: [ "/s/tempC", 7 ],    TS: 0("2013-07-10T10:04:12.391"),    L: 86400,    G: hmac_sha256 },    V: h’f89947160c73601c7a65cb5e088120266d0f0565160e3ff7d3907441cdf44cc9'    CAI: [ "/s/tempC", 1 ],    TS: 0("2013-07-10T10:04:12.855"),    L: 86400 }

Figure 7: Example Ticket Transfer Message

3.8. DTLS Channel Setup Between C and S

Using the information contained in a positive response to its Access Request (i.e. a Ticket Transfer message that contains a Face and a Client Information), C can initiate establishment of a new DTLS channel with S. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], with the following additional requirements:
1. C sets the psk_identity field of the ClientKeyExchange message to the ticket Face received in the Ticket Transfer message.

2. C uses the ticket Verifier as PSK when constructing the premaster secret.

Note1: As S cannot provide C with a meaningful PSK identity hint in response to C’s ClientHello message, S SHOULD NOT send a ServerKeyExchange message.

Note2: According to [RFC7252], CoAP implementations MUST support the ciphersuite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]. C is therefore expected to offer at least this ciphersuite to S.

Note3: The ticket is constructed by SAM such that S can derive the authorization information as well as the PSK (refer to Section 6 for details).

3.9. Authorized Resource Request Message

If the Client Information in the Ticket Transfer message contains CAI, C MUST ensure that it only sends requests that according to them are allowed. C therefore MUST check CAI, L and T before every request. If CAI is no longer valid according to L, C MUST terminate the DTLS connection with S and re-request the CAI from CAM using an Access Request Message.

On the Server side, successful establishment of the DTLS channel between C and S ties the SAM authorization information contained in the psk_identity field to this channel. Any request that S receives on this channel is checked against these authorization rules. Incoming CoAP requests that are not Authorized Resource Requests MUST be rejected by S with 4.01 response as described in Section 3.2.

S SHOULD treat an incoming CoAP request as Authorized Resource Request if the following holds:

1. The message was received on a secure channel that has been established using the procedure defined in Section 3.8.

2. The authorization information tied to the secure channel is valid.

3. The request is destined for S.

4. The resource URI specified in the request is covered by the authorization information.
5. The request method is an authorized action on the resource with respect to the authorization information.

Note that the authorization information is not restricted to a single resource URI. For example, role-based authorization can be used to authorize a collection of semantically connected resources simultaneously. Implicit authorization also provides access rights to authenticated clients for all actions on all resources that S offers. As a result, C can use the same DTLS channel not only for subsequent requests for the same resource (e.g. for block-wise transfer as defined in [I-D.ietf-core-block] or refreshing observe-relationships [I-D.ietf-core-observe]) but also for requests to distinct resources.

Incoming CoAP requests received on a secure channel according to the procedure defined in Section 3.8 MUST be rejected

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and

2. with response code 4.05 (Method Not Allowed) when the resource URI specified in the request covered by the authorization information but not the requested action.

Since SAM may limit the set of requested actions in its Ticket Grant message, C cannot know a priori if an Authorized Resource Request will succeed.

3.10. Dynamic Update of Authorization Information

Once a security association exists between a Client and a Resource Server, the Client can update the Authorization Information stored at the Server at any time. To do so, the Client creates a new Access Request for the intended action on the respective resource and sends this request to its CAM which checks and relays this request to the Server’s SAM as described in Section 3.4.

Note: Requesting a new Access Ticket also can be a Client’s reaction on a 4.03 or 4.05 error that it has received in response to an Authorized Resource Request.

Figure 8 depicts the message flow where C requests a new Access Tickets after a security association between C and S has been established using this protocol.
Figure 8: Overview of Dynamic Update Operation

Processing the Ticket Request is done at the SAM as specified in Section 3.6, i.e. the SAM checks whether or not the requested operation is permitted by the Resource Principal’s policy, and then return a Ticket Grant message with the result of this check. If access is granted, the Ticket Grant message contains an Access Ticket comprised of a public Ticket Face and a private Ticket Verifier. This authorization payload is relayed by CAM to the Client in a Ticket Transfer Message as defined in Section 3.7.

The major difference between dynamic update of Authorization Information and the initial handshake is the handling of a Ticket Transfer message by the Client that is described in Section 3.10.1.

3.10.1. Handling of Ticket Transfer Messages

If the security association with S still exists and S has indicated support for session renegotiation according to [RFC5746], the ticket Face SHOULD be used to renegotiate the existing DTLS session. In this case, the ticket Face is used as psk_identity as defined in Section 3.8. Otherwise, the Client MUST perform a new DTLS handshake according to Section 3.8 that replaces the existing DTLS session.

After successful completion of the DTLS handshake S updates the existing SAM Authorization Information for C according to the contents of the ticket Face.
Note: No mutual authentication between C and S is required for dynamic updates when a DTLS channel exists that has been established as defined in Section 3.8. S only needs to verify the authenticity and integrity of the ticket Face issued by SAM which is achieved by having performed a successful DTLS handshake with the ticket Face as psk_identity. This could even be done within the existing DTLS session by tunneling a CoDTLS [I-D.schmertmann-dice-codtls] handshake.

4. Ticket

Access tokens in DCAF are tickets that consist of two parts, namely the Face and the Client Information. The Face goes to S, the CI goes to the Client. The Face and the CI are parts of the same ticket.

S only needs the information contained in the Ticket Face to authorize the client and make sure that SAM generated the Ticket Face (S cannot make authorization decisions by itself and hence needs SAM to do it). No additional information about the Client is needed. S keeps the Ticket Face as long as it is valid.

4.1. Face

Face is the part of the ticket generated for S. Face MUST contain all information needed for authorized access to a resource:

- SAM Authorization Information
- A timestamp generated by SAM

Optionally, Face MAY also contain:

- A lifetime (optional)
- A DTLS pre-shared key (optional)

S MUST verify the integrity of Face, i.e. the information contained in Face stems from SAM and was not manipulated by anyone else.

Face MUST contain a timestamp to verify that the contained information is fresh. As constrained devices may not have a clock, timestamps MAY be generated using the clock ticks since the last reboot. To circumvent synchronization problems the timestamp MAY be generated by S and included in the first SAM Information message. Alternatively, SAM MAY generate the timestamp. In this case, SAM and S MUST use a time synchronization mechanism to make sure that S interprets the timestamp correctly.
Face MAY be encrypted. If Face contains a DTLS PSK, the whole content of Face MUST be encrypted.

The integrity of Face can be ensured by various means. Face may be encrypted by SAM with a key it shares with S. Alternatively, S can use a mechanism to generate the DTLS PSK which includes Face. S generates the key from the Face it received. The correct key can only be calculated with the correct Face (refer to Section 6 for details).

4.2. CI

The CI part of the ticket is generated for C. It contains the Verifier, i.e. the DTLS PSK for C and MAY contain Client Authorization Information generated by CAM to provide the COP’s authorization policies to C. The Verifier MUST NOT be transmitted over insecure channels.

4.3. Revocation

The existence of access tickets SHOULD be limited in time to avoid stale tickets that waste resources on S and C. This can be achieved either by explicit Revocation Messages to invalidate a ticket or implicitly by attaching a lifetime to the ticket.

4.3.1. Lifetime

Tickets MAY have a lifetime. SAM is responsible for defining the ticket lifetime. If SAM sets a lifetime for a ticket, SAM and S MUST use a time synchronization method to ensure that S is able to interpret the lifetime correctly. S SHOULD end the DTLS connection to C if the lifetime of a ticket has run out and it MUST NOT accept new requests. S MUST NOT accept tickets with an invalid lifetime.

If CAM provides CAI in the CI part of the ticket, CAM MAY add a lifetime for this CAI. If CI contains a lifetime, CAM and C MUST use a time synchronization method to ensure that C is able to interpret the lifetime correctly. C SHOULD end the DTLS connection to S and MUST NOT send new requests if the CAI in the ticket is no longer valid. C MUST NOT accept tickets with an invalid lifetime.

Note: Defining reasonable ticket lifetimes is difficult to accomplish. How long a client needs to access a resource depends heavily on the application scenario and may be difficult to decide for SAM.
4.3.2. Revocation Messages

SAM MAY revoke tickets by sending a ticket revocation message to S. If S receives a ticket revocation message, it MUST end the DTLS connection to C and MUST NOT accept any further requests from C.

If ticket revocation messages are used, S MUST check regularly if SAM is still available. If S cannot contact SAM, it MUST end all DTLS connections and reject any further requests from C.

Note: The loss of the connection between S and SAM prevents all access to S. This might especially be a severe problem if SAM is responsible for several Servers or even a whole network.

5. Payload Format and Encoding (application/dcaf+cbor)

Various messages types of the DCAF protocol carry payloads to express authorization information and parameters for generating the DTLS PSK to be used by C and S. In this section, a representation in Concise Binary Object Representation (CBOR, [RFC7049]) is defined.

DCAF data structures are defined as CBOR maps that contain key value pairs. For efficient encoding, the keys defined in this document are represented as unsigned integers in CBOR, i.e., major type 0. For improved reading, we use symbolic identifiers to represent the corresponding encoded values as defined in Table 1.
### Table 1: DCAF field identifiers encoded in CBOR

<table>
<thead>
<tr>
<th>Encoded Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000_00000</td>
<td>SAM</td>
</tr>
<tr>
<td>0b000_00001</td>
<td>SAI</td>
</tr>
<tr>
<td>0b000_00010</td>
<td>CAI</td>
</tr>
<tr>
<td>0b000_00011</td>
<td>E</td>
</tr>
<tr>
<td>0b000_00100</td>
<td>K</td>
</tr>
<tr>
<td>0b000_00101</td>
<td>TS</td>
</tr>
<tr>
<td>0b000_00110</td>
<td>L</td>
</tr>
<tr>
<td>0b000_00111</td>
<td>G</td>
</tr>
<tr>
<td>0b000_01000</td>
<td>F</td>
</tr>
<tr>
<td>0b000_01001</td>
<td>V</td>
</tr>
</tbody>
</table>

The following list describes the semantics of the keys defined in DCAF.

**SAM:** Server Authorization Manager. This attribute denotes the Server Authorization Manager that is in charge of the resource specified in attribute R. The attribute’s value is a string that contains an absolute URI according to Section 4.3 of [RFC3986].

**SAI:** SAM Authorization Information. A data structure used to convey authorization information from SAM to S. It describes C’s permissions for S according to SAM, e.g., which actions C is allowed to perform on an R of S. The SAI attribute contains an AIF object as defined in [I-D.bormann-core-ace-aif]. C uses SAI for its Access Request messages.

**CAI:** CAM Authorization Information. A data structure used to convey authorization information from CAM to C. It describes the C’s permissions for S according to CAM, e.g., which actions C is allowed to perform on an R of S. The CAI attribute contains an AIF object as defined in [I-D.bormann-core-ace-aif].
E: Encrypted Ticket Face. A binary string containing an encrypted ticket Face.

K: Key. A string that identifies the shared key between S and SAM that can be used to decrypt the contents of E. If the attribute E is present and no attribute K has been specified, the default is to use the current session key for the secured channel between S and SAM.

TS: Time Stamp. An optional time stamp that indicates the instant when the access ticket request was formed. This attribute can be used by the Server in an SAM Information message to convey a time stamp in its local time scale (e.g. when it does not have a real time clock with synchronized global time). When the attribute’s value is encoded as a string, it MUST contain a valid UTC timestamp without time zone information. When encoded as integer, TS contains a system timestamp relative to the local time scale of its generator, usually S.

L: Lifetime. A lifetime of the ticket. When encoded as a string, L MUST denote the ticket’s expiry time as a valid UTC timestamp without time zone information. When encoded as an integer, L MUST denote the ticket’s validity period in seconds relative to TS.

G: DTLS PSK Generation Method. A numeric identifier for the method that S MUST use to derive the DTLS PSK from the ticket Face. This attribute MUST NOT be used when attribute V is present within the contents of F. This specification uses symbolic identifiers for improved readability. The corresponding numeric values encoded in CBOR are defined in Table 2. A registry for these codes is defined in Section 13.1.

F: Ticket Face. An object containing the fields SAI, TS, and optionally G, L and V.

V: Ticket Verifier. A binary string containing the shared secret between C and S.
Table 2: CBOR encoding for DTLS PSK Key Generation Methods

<table>
<thead>
<tr>
<th>Encoded Value</th>
<th>Mnemonic</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000_00000</td>
<td>hmac_sha256</td>
<td>mandatory</td>
</tr>
<tr>
<td>0b000_00001</td>
<td>hmac_sha384</td>
<td>optional</td>
</tr>
<tr>
<td>0b000_00010</td>
<td>hmac_sha512</td>
<td>optional</td>
</tr>
</tbody>
</table>

5.1. Examples

The following example specifies a SAM that will be accessed using HTTP over TLS. The request URI is set to "/a?ep=%5B2001:DB8::dcaf:1234%5D" (hence denoting the endpoint address to authorize). TS denotes a local timestamp in UTC.

POST /a?ep=%5B2001:DB8::dcaf:1234%5D HTTP/1.1
Host: sam.example.com
Content-Type: application/dcaf+cbor
{SAM: "https://sam.example.com/a?ep=%5B2001:DB8::dcaf:1234%5D",
SAI: ["coaps://temp451.example.com/s/tempC", 1],
TS: 0("2013-07-14T11:58:22.923")}

The following example shows a ticket for the distributed key generation method (cf. Section 6.2), comprised of a Face (F) and a Verifier (V). The Face data structure contains authorization information SAI, a client descriptor, a timestamp using the local time scale of S, and a lifetime relative to S’s time scale.

The DTLS PSK Generation Method is set to hmac_sha256 denoting that the distributed key derivation is used as defined in Section 6.2 with SHA-256 as HMAC function.

The Verifier V contains a shared secret to be used as DTLS PSK between C and S.
HTTP/1.1 200 OK
Content-Type: application/dcaf+cbor
{
    F: {
        SAI: [ "/s/tempC", 1 ],
        TS: 2938749,
        L:  3600,
        G: hmac_sha256
    },
    V: h'48ae5a81b87241d81618f56cab0b65ec
        441202f81faabbe10075b20cb57fa939'
}
The Face may be encrypted as illustrated in the following example. Here, the field E carries an encrypted Face data structure that contains the same information as the previous example, and an additional Verifier. Encryption was done with a secret shared by SAM and S. (This example uses AES128_CCM with the secret { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f } and S’s timestamp { 0x00, 0x2C, 0xD7, 0x7D } as nonce.) Line breaks have been inserted to improve readability.

The attribute K describes the identity of the key to be used by S to decrypt the contents of attribute E. Here, The value "key0" in this example is used to indicate that the shared session key between S and SAM was used for encrypting E.

{
    E: h'2e75eeae01b831e0b65c2976e06d90f4
        82135bec5efef3be3d31520b2fa8c6fb
        f572f817203bf7a0940bb6183697567c
        e291b03e9fca5e9cbdfa7e560322d4ed
        3a659f44a542e55331a1a9f43d7f',
    K: "key0",
    V: h'48ae5a81b87241d81618f56cab0b65ec
        441202f81faabbe10075b20cb57fa939'
}
The decrypted contents of E are depicted below (whitespace has been added to improve readability). The presence of the attribute V indicates that the DTLS PSK Transfer is used to convey the session key (cf. Section 6.1).
6.  DTLS PSK Generation Methods

One goal of the DCAF protocol is to provide for a DTLS PSK shared between C and S. SAM and S MUST negotiate the method for the DTLS PSK generation.

6.1.  DTLS PSK Transfer

The DTLS PSK is generated by AS and transmitted to C and S using a secure channel.

The DTLS PSK transfer method is defined as follows:

- SAM generates the DTLS PSK using an algorithm of its choice.
- SAM MUST include a representation of the DTLS PSK in Face and encrypt it together with all other information in Face with a key $K(\text{SAM},S)$ it shares with S. How SAM and S exchange $K(\text{SAM},S)$ is not in the scope of this document. SAM and S MAY use their preshared key as $K(\text{SAM},S)$.
- SAM MUST include a representation of the DTLS PSK in the Verifier.
- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- S MUST decrypt Face using $K(\text{SAM},S)$

6.2.  Distributed Key Derivation

SAM generates a DTLS PSK for C which is transmitted using a secure channel. S generates its own version of the DTLS PSK using the information contained in Face (see also Section 4.1).

The distributed key derivation method is defined as follows:
o SAM and S both generate the DTLS PSK using the information included in Face. They use an HMAC algorithm on Face with a shared key K(SAM,S). The result serves as the DTLS PSK. How SAM and S exchange K(SAM,S) is not in the scope of this document. They MAY use their preshared key as K(SAM,S). How SAM and S negotiate the used HMAC algorithm is also not in the scope of this document. They MAY however use the HMAC algorithm they use for their DTLS connection.

o SAM MUST include a representation of the DTLS PSK in the Verifier.

o As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.

o SAM MUST NOT include a representation of the DTLS PSK in Face.

o SAM MUST NOT encrypt Face.

7. Authorization Configuration

For the protocol defined in this document, proper configuration of CAM and SAM is crucial. The principals that are in charge of the resource, S and SAM, and the principals that are in charge of C and CAM need to define the respective permissions. The data representation of these permissions are not in the scope of this document.

8. Trust Relationships

C has a trust relationship with CAM: C trusts CAM to act in behalf of C’s principal. S has a trust relationship with SAM: S trusts SAM to act in behalf of S’s principal.

Obviously, CAM trusts C with the specific permissions it hands over to it. How this trust is established, is not in the scope of this document. It may be achieved by using a bootstrapping mechanism similar to [bergmann12].

Additionally, SAM and CAM need to have a trust relationship established. Its establishment is also not in the scope of this document. It fulfills the following conditions:

1. SAM has means to authenticate CAM (e.g. it has a certificate of CAM or a PKI in which CAM is included) and vice versa

2. As far as SAM needs to rely on the different clients of CAM to receive different permissions, it can be sure that CAM correctly identifies these clients towards SAM and does not leak tickets
that have been generated for a specific client C to another client.

SAM trusts C indirectly because it trusts CAM and CAM vouches for C. The DCAF Protocol does not provide any means for SAM to validate that a resource request stems from C.

C indirectly trusts SAM with some potentially confidential information, and that SAM correctly represents S, because CAM trusts SAM.

CAM trusts S indirectly because it trusts SAM and SAM vouches for S.

C implicitly trusts S with some potentially confidential information because it trusts CAM and because S can prove that it shares a key with SAM.

CAM <------------------> SAM
/|
/ |\                      /|\
\ /                      \ /\ C .....................  S
\/                        \/


CoAP utilizes the Web Linking format [RFC5988] to facilitate discovery of services in an M2M environment. [RFC6690] defines specific link parameters that can be used to describe resources to be listed in a resource directory [I-D.ietf-core-resource-directory].

9.1. The "auth-request" Link Relation

This section defines a resource type "auth-request" that can be used by clients to retrieve the request URI for a server’s authorization service. When used with the parameter rt in a web link, "auth-request" indicates that the corresponding target URI can be used in a POST message to request authorization for the resource and action that are described in the request payload.

The Content-Format "application/dcaf+cbor with numeric identifier TBD1 defined in this specification MAY be used to express access requests and their responses.
The following example shows the web link used by CAM in this document to relay incoming Authorization Request messages to SAM. (Whitespace is included only for readability.)

```
<client-authorize>;rt="auth-request";ct=TBD1
 ;title="Contact Remote Authorization Manager"
```

The resource directory that hosts the resource descriptions of S could list the following description. In this example, the URI "ep/node138/a/switch2941" is relative to the resource context "coaps://sam.example.com/", i.e. the Server Authorization Manager SAM.

```
<ep/node138/a/switch2941>;rt="auth-request";ct=TBD1;ep="node138"
 ;title="Request Client Authorization"
 ;anchor="coaps://sam.example.com/"
```

10. Examples

This section gives a number of short examples with message flows for the initial Unauthorized Resource Request and the subsequent retrieval of a ticket from SAM. The notation here follows the actors conventions defined in Section 1.2.1. The payload format is encoded as proposed in Section 5. The IP address of SAM is 2001:DB8::1, the IP address of S is 2001:DB8::dcaf:1234, and C’s IP address is 2001:DB8::c.

10.1. Access Granted

This example shows an Unauthorized PUT request from C to S that is answered with a SAM Information message. C then sends a POST request to CAM with a description of its intended request. CAM forwards this request to SAM using CoAP over a DTLS-secured channel. The response from SAM contains an access ticket that is relayed back to CAM.

```
C --&gt; S
PUT a/switch2941 [Mid=1234]
Content-Format: application/senml+json
{"e": [{"bv": "1"}]}
```

```
C &lt;-&gt; S
4.01 Unauthorized [Mid=1234]
Content-Format: application/dcaf+cbor
{SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941"}
```

```
C --&gt; CAM
POST client-authorize [Mid=1235,Token="tok"]
Content-Format: application/dcaf+cbor
```

CAM --+ SAM [Mid=23146]
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor

CAM <-- SAM
2.05 Content [Mid=23146]
Content-Format: application/dcaf+cbor

C <--> CAM
2.05 Content [Mid=1235, Token="tok"]
Content-Format: application/dcaf+cbor

C --> S
ClientHello (TLS_PSK_WITH_AES_128_CCM_8)

C <--> S
ServerHello (TLS_PSK_WITH_AES_128_CCM_8)
ServerHelloDone

C <--> S
ClientKeyExchange
  psk_identity=0xa301826c612f73776974636832393431
  0x0505c077323031332d30372d30345432

0x303a31373a33382e3030320700

(C decodes the contents of V and uses the result as PSK)
ChangeCipherSpec
Finished

(S calculates PSK from SAI, TS and its session key
HMAC_sha256(0xa301826c612f73776974636832393431
  0x0505c077323031332d30372d30345432
  0x303a31373a33382e3030320700,
  0x736563726574)
  = 0x7ba4d9e287c8...
)

C <-- S
ChangeCipherSpec
Finished

10.2. Access Denied

This example shows a denied Authorization request for the DELETE operation.
C --> S
DELETE a/switch2941

C <-- S
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941"
}

C --> CAM
POST client-authorize
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 8]
}

CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 8]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor

C <-- CAM
2.05 Content
Content-Format: application/dcaf+cbor

10.3. Access Restricted

This example shows a denied Authorization request for the operations GET, PUT, and DELETE. SAM grants access for PUT only.
10.4. Implicit Authorization

This example shows an Authorization request using implicit authorization. CAM initially requests the actions GET and POST on the resource "coaps://[2001:DB8::dcaf:1234]/a/switch2941". SAM returns a ticket that has no SAI field in its ticket Face, hence implicitly authorizing C.

CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
    SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
    SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 13]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor
{ F: {
    SAI: ["a/switch2941", 5],
    TS: 0("2013-07-04T21:33:11.930"),
    G: hmac_sha256
},
V: h’c7b5774f2ddcb548f4ad74b30a1b2e5b6b04e66a9995edd2545e5a06216c53d’
}

11. Specific Usage Scenarios

The general DCAF architecture outlined in Section 3.1 illustrates the various actors who participate in the message exchange for authenticated authorization. The message types defined in this document cover the most general case where all four actors are separate entities that may or may not reside on the same device.

Special implementation considerations apply when one single entity takes the role of more than one actor. This section gives advice on the most common usage scenarios where the Client Authorization Manager and Client, the Server Authorization Manager and Server or both Authorization Managers reside on the same (less-constrained) device and have a means of secure communication outside the scope of this document.

11.1. Combined Authorization Manager and Client

When CAM and C reside on the same (less-constrained) device, the Access Request and Ticket Transfer messages can be substituted by other means of secure communication. Figure 9 shows a simplified message exchange for a combined CAM+C device.

```
CAM+C             S             SAM
|                | <== DTLS chan. ==>  |
| [Resource Req.--->] |                |
<------ TLS/DTLS chan. (Mutual Auth) ===> |
| Ticket Request --------------->
| <---------------------    Ticket Grant |
| <<= DTLS chan. ==>
| Auth. Res. Req. ->
```

Figure 9: Combined Client Authorization Manager and Client

11.1.1. Creating the Ticket Request Message

When CAM+C receives an SAM Information message as a reaction to an Unauthorized Request message, it creates a Ticket Request message as follows:
1. The destination of the Ticket Request message is derived from the authority information in the URI contained in field "SAM" of the SAM Information message payload.

2. The request method is POST.

3. The request URI is constructed from the SAM field received in the SAM Information message payload.

4. The payload contains the SAM field from the SAM Information message, an absolute URI of the resource that CAM+C wants to access, the actions that CAM+C wants to perform on the resource, and any time stamp generated by S that was transferred with the SAM Information message.

5. A label that describes CAM+C is added to the payload.

11.1.2. Processing the Ticket Grant Message

Based on the Ticket Grant message, CAM+C is able to establish a DTLS channel with S. To do so, CAM+C sets the psk_identity field of the DTLS ClientKeyExchange message to the ticket Face received in the Ticket Grant message and uses the ticket Verifier as PSK when constructing the premaster secret.

11.2. Combined Client Authorization Manager and Server Authorization Manager

In certain scenarios, CAM and SAM may be combined to a single entity that knows both, C and S, and decides if their actions are authorized. Therefore, no explicit communication between CAM and SAM is necessary, resulting in omission of the Ticket Request and Ticket Grant messages. Figure 10 depicts the resulting message sequence in this simplified architecture.
11.2.1. Processing the Access Request Message

When receiving an Access Request message, CAM+SAM performs the checks specified in Section 3.5 and returns a 4.00 (Bad Request) response in case of failure. Otherwise, if the checks have succeeded, CAM+SAM evaluates the contents of Access Request message as described in Section 3.6.

The decision on the access request is performed by CAM+SAM with respect to the stored policies. When the requested action is permitted on the respective resource, CAM+SAM generates an access ticket as outlined in Section 4.1 and creates a Ticket Transfer message to convey the access ticket to the Client.

11.2.2. Creating the Ticket Transfer Message

A Ticket Transfer message is constructed as a 2.05 response with the access ticket contained in its payload. The response MAY contain a Max-Age option to indicate the ticket’s lifetime to the receiving Client.

This specification defines a CBOR data representation for the access ticket as illustrated in Section 3.6.

11.3. Combined Server Authorization Manager and Server

If SAM and S are colocated in one entity (SAM+S), the main objective is to allow CAM to delegate access to C. Accordingly, the authorization information could be replaced by a nonce internal to SAM+S. (TBD.)
12. Security Considerations

As this protocol builds on transitive trust between Authorization Managers as mentioned in Section 8, SAM has no direct means to validate that a resource request originates from C. It has to trust CAM that it correctly vouches for C and that it does not give authorization tickets meant for C to another client nor disclose the contained session key.

The Authorization Managers also could constitute a single point of failure. If the Server Authorization Manager fails, the resources on all Servers it is responsible for cannot be accessed any more. If a Client Authorization Manager fails, all clients it is responsible are not able to access resources on a Server. Thus, it is crucial for large networks to use Authorization Managers in a redundant setup.

13. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification.
13.1. DTLS PSK Key Generation Methods

A sub-registry for the values indicating the PSK key generation method as contents of the field G in a payload of type application/dcaf+cbor is defined. Values in this sub-registry are numeric integers encoded in Concise Binary Object Notation (CBOR, [RFC7049]). This document follows the notation of [RFC7049] for binary values, i.e. a number starts with the prefix "0b". The major type is separated from the actual numeric value by an underscore to emphasize the value’s internal structure.

Initial entries in this sub-registry are as follows:

<table>
<thead>
<tr>
<th>Encoded Value</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000_00000</td>
<td>hmac_sha256</td>
<td>[RFC-XXXX]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b000_00001</td>
<td>hmac_sha384</td>
<td>[RFC-XXXX]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b000_00010</td>
<td>hmac_sha512</td>
<td>[RFC-XXXX]</td>
</tr>
</tbody>
</table>

Table 3: DTLS PSK Key Generation Methods

New methods can be added to this registry based on designated expert review according to [RFC5226].

(TBD: criteria for expert review.)

13.2. dcaf+cbor Media Type Registration

Type name: application

Subtype name: dcaf+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7049]. Specifically, only the primitive data types String and Number are allowed. The type Number is restricted to unsigned integers (i.e., no negative numbers, fractions or exponents are allowed). Encoding MUST be UTF-8. These restrictions simplify implementations on devices that have very limited memory capacity.
Security considerations: TBD

Interoperability considerations: TBD

Published specification: [RFC-XXXX]

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): dcaf

Macintosh file type code(s): none

Person & email address to contact for further information: TBD

Intended usage: COMMON

Restrictions on usage: None

Author: TBD

Change controller: IESG

13.3. CoAP Content Format Registration

This document specifies a new media type application/dcaf+cbor (cf. Section 13.2). For use with CoAP, a numeric Content-Format identifier is to be registered in the "CoAP Content-Formats" sub-registry within the "CoRE Parameters" registry.

Note to RFC Editor: Please replace all occurrences of "RFC-XXXX" with the RFC number of this specification.

+-----------------------+----------+------+------------+
| Media type            | Encoding | Id.  | Reference  |
| application/dcaf+cbor | -        | TBD1 | [RFC-XXXX] |

14. References

14.1. Normative References


14.2. Informative References


[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.

[I-D.ietf-core-resource-directory]

[I-D.schmertmann-dice-codtls]


[bergmann12]

Authors’ Addresses

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

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

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

Phone: +49-421-218-63904
Email: bergmann@tzi.org
ACE use cases
draft-ietf-ace-usecases-03

Abstract

Constrained devices are nodes with limited processing power, storage space and transmission capacities. These devices in many cases do not provide user interfaces and are often intended to interact without human intervention.

This document comprises a collection of representative use cases for the application of authentication and authorization in constrained environments. These use cases aim at identifying authorization problems that arise during the lifecycle of a constrained device and are intended to provide a guideline for developing a comprehensive authentication and authorization solution for this class of scenarios.

Where specific details are relevant, it is assumed that the devices use the Constrained Application Protocol (CoAP) as communication protocol, however most conclusions apply generally.

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."
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 September 10, 2015.

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 ......................................................... 3
   1.1.  Terminology .................................................. 4
2.  Use Cases ........................................................... 4
   2.1.  Container monitoring ......................................... 4
         2.1.1.  Bananas for Munich .................................... 5
         2.1.2.  Authorization Problems Summary ........................ 6
   2.2.  Home Automation ............................................... 6
         2.2.1.  Controlling the Smart Home Infrastructure ............ 7
         2.2.2.  Seamless Authorization .................................. 7
         2.2.3.  Remotely letting in a visitor .......................... 7
         2.2.4.  Authorization Problems Summary ........................ 8
   2.3.  Personal Health Monitoring .................................... 9
         2.3.1.  John and the heart rate monitor ....................... 9
         2.3.2.  Authorization Problems Summary ........................ 10
   2.4.  Building Automation .......................................... 11
         2.4.1.  Device Lifecycle ........................................ 11
         2.4.2.  Authorization Problems Summary ........................ 13
   2.5.  Smart Metering ................................................ 14
         2.5.1.  Drive-by metering ....................................... 14
         2.5.2.  Meshed Topology .......................................... 15
         2.5.3.  Advanced Metering Infrastructure ....................... 15
         2.5.4.  Authorization Problems Summary ........................ 16
   2.6.  Sports and Entertainment ..................................... 16
         2.6.1.  Dynamically Connecting Smart Sports Equipment .......... 17
         2.6.2.  Authorization Problems Summary ........................ 17
   2.7.  Industrial Control Systems ................................... 18
1. Introduction

Constrained devices [RFC7228] are nodes with limited processing power, storage space and transmission capacities. These devices are often battery-powered and in many cases do not provide user interfaces.

Constrained devices benefit from being interconnected using Internet protocols. However, due to the devices’ limitations, commonly used security protocols are not always easily applicable. As the devices are expected to be integrated in all aspects of everyday life, the application of adequate security mechanisms is required to prevent attackers from gaining control over data or functions important to our lives.

This document comprises a collection of representative use cases for the application of authentication and authorization in constrained environments. These use cases aim at identifying authorization problems that arise during the lifecycle of a constrained device. Note that this document does not aim at collecting all possible use cases.

We assume that the communication between the devices is based on the Representational State Transfer (REST) architectural style, i.e. a device acts as a server that offers resources such as sensor data and actuators. The resources can be accessed by clients, sometimes without human intervention (M2M). In some situations the communication will happen through intermediaries (e.g. gateways, proxies).

Where specific detail is necessary it is assumed that the devices communicate using CoAP [RFC7252], although most conclusions are generic.
1.1. Terminology

Readers are required to be familiar with the terms defined in [RFC7228]. In addition, this document uses the following terminology:

2. Use Cases

This section lists use cases involving constrained devices with certain authorization problems to be solved. Each use case first presents a general description of the application area, then one or more specific use cases, and finally a summary of the authorization-related problems users need to be solved.

There are various reasons for assigning a function (client or server) to a device, e.g. which device initiates the conversation, how do devices find each other, etc. The definition of the function of a device in a certain use case is not in scope of this document. Readers should be aware that there might be reasons for each setting and that endpoints might even have different functions at different times.

2.1. Container monitoring

The ability of sensors to communicate environmental data wirelessly opens up new application areas. The use of such sensor systems makes it possible to continuously track and transmit specific characteristics such as temperature, humidity and gas content during the transportation and storage of goods.

The proper handling of the sensors in this scenario is not easy to accomplish. They have to be associated to the appropriate pallet of the respective container. Moreover, the goods and the corresponding sensors belong to specific customers.

During the shipment to their destination the goods often pass stops where they are transloaded to other means of transportation, e.g. from ship transport to road transport.

The transportation and storage of perishable goods is especially challenging since they have to be stored at a constant temperature and with proper ventilation. Additionally, it is very important for the vendors to be informed about irregularities in the temperature and ventilation of fruits to avoid the delivery of decomposed fruits to their customers. The need for a constant monitoring of perishable goods has led to projects such as The Intelligent Container (http://www.intelligentcontainer.com).
2.1.1. Bananas for Munich

A fruit vendor grows bananas in Costa Rica for the German market. It instructs a transport company to deliver the goods via ship to Rotterdam where they are picked up by trucks and transported to a ripening facility. A Munich supermarket chain buys ripened bananas from the fruit vendor and transports them from the ripening facility to the individual markets with their own company trucks.

The fruit vendor’s quality management wants to assure the quality of their products and thus equips the banana boxes with sensors. The state of the goods is monitored consistently during shipment and ripening and abnormal sensor values are recorded. Additionally, the sensor values are used to control the climate within the cargo containers. The sensors therefore need to communicate with the climate control system. Since a wrong sensor value leads to a wrong temperature and thus to spoiled goods, the integrity of the sensor data must be assured. The banana boxes within a container will in most cases belong to the same owner. Adjacent containers might contain goods and sensors of different owners.

The personnel that transloads the goods must be able to locate the goods meant for a specific customer. However the fruit vendor does not want to disclose sensor information pertaining to the condition of the goods to other companies and therefore wants to assure the confidentiality of this data. Thus, the transloading personnel is only allowed to access logistic information. Moreover, the transloading personnel is only allowed to access the data for the time of the transloading.

Due to the high water content of the fruits, the propagation of radio waves is hindered, thus often inhibiting direct communication between nodes [Jedermann14]. Instead, messages are forwarded over multiple hops. The sensors in the banana boxes cannot always reach the Internet during the journey.

In the ripening facility bananas are stored until they are ready for selling. The banana box sensors are used to control the ventilation system and to monitor the degree of ripeness of the bananas. Ripe bananas need to be identified and sold before they spoil.

The supermarket chain gains ownership of the banana boxes when the bananas have ripened and are ready to leave the ripening facility.
2.1.2. Authorization Problems Summary

- U1.1 Fruit vendors, transloading personnel and container owners want to grant different authorizations for their resources and/or endpoints to different parties.

- U1.2 The fruit vendor requires the integrity of the sensor data that pertains the state of the goods for climate control and to ensure the quality of the monitored recordings.

- U1.3 The container owner requires the integrity of the sensor data that is used for climate control.

- U1.4 The fruit vendor requires the confidentiality of the sensor data that pertains the state of the goods.

- U1.5 The fruit vendor may have several types of data that may be controlled by the same endpoint, e.g., sensor data and the data used for logistics.

- U1.6 The fruit vendor requires the confidentiality of the data that is used to locate the goods.

- U1.7 The fruit vendor requires the integrity of the data that is used to locate the goods.

- U1.8 The transloading personnel requires the integrity of the data that is used to locate the goods.

- U1.9 The container owner and the fruit vendor may not be present at the time of access and cannot manually intervene in the authorization process.

- U1.10 The fruit vendor, container owner and transloading company want to grant temporary access permissions to a party.

- U1.11 Messages between client and resource server might need to be forwarded over multiple hops.

- U1.12 The constrained devices might not always be able to reach the Internet.

2.2. Home Automation

Automation of the home has the potential to become a big future market for the Internet of Things. A home automation system connects devices in a house to the Internet and thus makes them accessible and
manageable remotely. Such devices might control for example heating, ventilation, lighting, home entertainment or home security.

Such a system needs to accommodate a number of regular users (inhabitants, close friends, cleaning personnel) as well as a heterogeneous group of dynamically varying users (visitors, repairmen, delivery men).

As the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users.

2.2.1. Controlling the Smart Home Infrastructure

Alice and her husband Bob own a flat which is equipped with home automation devices such as HVAC and shutter control, and they have a motion sensor in the corridor which controls the light bulbs there.

Alice and Bob can control the shutters and the temperature in each room using either wall-mounted touch panels or an internet connected device (e.g. a smartphone). Since Alice and Bob both have a full-time job, they want to be able to change settings remotely, e.g. turn up the heating on a cold day if they will be home earlier than expected.

The couple does not want people in radio range of their devices, e.g. their neighbors, to be able to control them without authorization. Moreover, they don’t want burglars to be able to deduce behavioral patterns from eavesdropping on the network.

2.2.2. Seamless Authorization

Alice buys a new light bulb for the corridor and integrates it into the home network, i.e. makes resources known to other devices in the network. Alice makes sure that the new light bulb and her other devices in the network get to know the authorization policies for the new device. Bob is not at home, but Alice wants him to be able to control the new device with his devices (e.g. his smartphone) without the need for additional administration effort. She provides the necessary configurations for that.

2.2.3. Remotely letting in a visitor

Alice and Bob have equipped their home with automated connected door-locks and an alarm system at the door and the windows. The couple can control this system remotely.
Alice and Bob have invited Alice’s parents over for dinner, but are stuck in traffic and cannot arrive in time, while Alice’s parents who use the subway will arrive punctually. Alice calls her parents and offers to let them in remotely, so they can make themselves comfortable while waiting. Then Alice sets temporary permissions that allow them to open the door, and shut down the alarm. She wants these permissions to be only valid for the evening since she does not like it if her parents are able to enter the house as they see fit.

When Alice’s parents arrive at Alice’s and Bob’s home, they use their smartphone to communicate with the door-lock and alarm system.

2.2.4. Authorization Problems Summary

- U2.1 A home owner (Alice and Bob in the example above) wants to spontaneously provision authorization means to visitors.
- U2.2 A home owner wants to spontaneously change the home’s access control policies.
- U2.3 A home owner wants to apply different access rights for different users.
- U2.4 The home owners want to grant temporary access permissions to a party.
- U2.5 The smart home devices need to be able to communicate with different control devices (e.g. wall-mounted touch panels, smartphones, electronic key fobs).
- U2.6 The home owner wants to be able to configure authorization policies remotely.
- U2.7 Authorized Users want to be able to obtain access with little effort.
- U2.8 The owners of the automated home want to prevent unauthorized entities from being able to deduce behavioral profiles from devices in the home network.
- U2.9 Usability is particularly important in this scenario since the necessary authorization related tasks in the lifecycle of the device (commissioning, operation, maintenance and decommissioning) likely need to be performed by the home owners who in most cases have little knowledge of security.
Home Owners want their devices to seamlessly (and in some cases even unnoticeably) fulfill their purpose. The administration effort needs to be kept at a minimum.

2.3. Personal Health Monitoring

The use of wearable health monitoring technology is expected to grow strongly, as a multitude of novel devices are developed and marketed. The need for open industry standards to ensure interoperability between products has lead to initiatives such as Continua Alliance (continuaalliance.org) and Personal Connected Health Alliance (pchalliance.org). Personal health devices are typically battery driven, and located physically on the user. They monitor some bodily function, such as e.g. temperature, blood pressure, or pulse. They are connected to the Internet through an intermediary base-station, using wireless technologies. Through this connection they report the monitored data to some entity, which may either be the user herself, or some medical personnel in charge of the user.

Medical data has always been considered as very sensitive, and therefore requires good protection against unauthorized disclosure. A frequent, conflicting requirement is the capability for medical personnel to gain emergency access, even if no specific access rights exist. As a result, the importance of secure audit logs increases in such scenarios.

Since the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users. Parts of the system must operate with minimal maintenance. Especially frequent changes of battery are unacceptable.

2.3.1. John and the heart rate monitor

John has a heart condition, that can result in sudden cardiac arrests. He therefore uses a device called HeartGuard that monitors his heart rate and his position. In case of a cardiac arrest it automatically sends an alarm to an emergency service, transmitting John’s current location. This requires the device to be close to a wireless access point, in order to be able to get an Internet connection (e.g. John’s smartphone).

The device includes some authentication mechanism, in order to prevent other persons who get physical access to it from acting as the owner and messing up the access control and security settings.

John can configure additional persons that get notified in an emergency, for example his daughter Jill. Furthermore the device
stores data on John’s heart rate, which can later be accessed by a physician to assess the condition of John’s heart.

However John is a privacy conscious person, and is worried that Jill might use HeartGuard to monitor his location while there is no emergency. Furthermore he doesn’t want his health insurance to get access to the HeartGuard data, or even to the fact that he is wearing a HeartGuard, since they might refuse to renew his insurance if they decided he was too big a risk for them.

Finally John, while being comfortable with modern technology and able to operate it reasonably well, is not trained in computer security. He therefore needs an interface for the configuration of the HeartGuard security that is easy to understand and use. If John does not understand the meaning of a setting, he tends to leave it alone, assuming that the manufacturer has initialized the device to secure settings.

NOTE: Monitoring of some state parameter (e.g. an alarm button) and the position of a person also fits well into an elderly care service. This is particularly useful for people suffering from dementia, where the relatives or caregivers need to be notified of the whereabouts of the person under certain conditions. In this case it is not the patient that decides about access.

2.3.2. Authorization Problems Summary

- **U3.1** The wearer of an eHealth device (John in the example above) wants to pre-configure special access rights in the context of an emergency.

- **U3.2** The wearer of an eHealth device wants to selectively allow different persons or groups access to medical data.

- **U3.3** The Security measures could affect battery lifetime of the device and changing the battery is very inconvenient.

- **U3.4** Devices are often used with default access control settings.

- **U3.5** Wearers of eHealth devices are often not trained in computer use, and especially computer security.

- **U3.6** Security mechanisms themselves could provide opportunities for denial of service attacks on the device.

- **U3.7** The device provides a service that can be fatal for the wearer if it fails. Accordingly, the wearer wants a security mechanism to provide a high level of security.
2.4. Building Automation

Buildings for commercial use such as shopping malls or office buildings nowadays are equipped increasingly with semi-automatic components to enhance the overall living quality and to save energy where possible. This includes for example heating, ventilation and air condition (HVAC) as well as illumination and security systems such as fire alarms.

Different areas of these buildings are often exclusively leased to different companies. However they also share some of the common areas of the building. Accordingly, a company must be able to control the light and HVAC system of its own part of the building and must not have access to control rooms that belong to other companies.

Some parts of the building automation system such as entrance illumination and fire alarm systems are controlled either by all parties together or by a service company.

2.4.1. Device Lifecycle

2.4.1.1. Installation and Commissioning

A building is hired out to different companies for office space. This building features various automated systems, such as a fire alarm system, which is triggered by several smoke detectors which are spread out across the building. It also has automated HVAC, lighting and physical access control systems.

A vacant area of the building has been recently leased to company A. Before moving into its new office, Company A wishes to replace the lighting with a more energy efficient and a better light quality luminaries. They hire an installation and commissioning company C to redo the illumination. Company C is instructed to integrate the new lighting devices, which may be from multiple manufacturers, into the existing lighting infrastructure of the building which includes presence sensors, switches, controllers etc.

Company C gets the necessary authorization from the service company to interact with the existing Building and Lighting Management System (BLMS). To prevent disturbance to other occupants of the building, Company C is provided authorization to perform the commissioning only during non-office hours and only to modify configuration on devices belonging to the domain of Company A’s space. After installation (wiring) of the new lighting devices, the commissioner adds the devices into the company A’s lighting domain.
Once the devices are in the correct domain, the commissioner authorizes the interaction rules between the new lighting devices and existing devices like presence sensors. For this, the commissioner creates the authorization rules on the BLMS which define which lights form a group and which sensors/switches/controllers are allowed to control which groups. These authorization rules may be context based like time of the day (office or non-office hours) or location of the handheld lighting controller etc.

2.4.1.2. Operational

Company A’s staff move into the newly furnished office space. Most lighting is controlled by presence sensors which control the lighting of specific group of lights based on the authorization rules in the BLMS. Additionally employees are allowed to manually override the lighting brightness and color in their office by using the switches or handheld controllers. Such changes are allowed only if the authorization rules exist in the BLMS. For example lighting in the corridors may not be manually adjustable.

At the end of the day, lighting is dimmed down or switched off if no occupancy is detected even if manually overridden during the day.

On a later date company B also moves into the same building, and shares some of the common spaces with company A. On a really hot day James who works for company A turns on the air condition in his office. Lucy who works for company B wants to make tea using an electric kettle. After she turned it on she goes outside to talk to a colleague until the water is boiling. Unfortunately, her kettle has a malfunction which causes overheating and results in a smoldering fire of the kettle’s plastic case.

Due to the smoke coming from the kettle the fire alarm is triggered. Alarm sirens throughout the building are switched on simultaneously (using a broadcast or multicast) to alert the staff of both companies. Additionally, the ventilation system of the whole building is closed off to prevent the smoke from spreading and to withdraw oxygen from the fire. The smoke cannot get into James’ office although he turned on his air condition because the fire alarm overrides the manual setting by sending commands (broadcast or multicast) to switch off all the air conditioning.

The fire department is notified of the fire automatically and arrives within a short time. After inspecting the damage and extinguishing the smoldering fire a fire fighter resets the fire alarm because only the fire department is authorized to do that.
2.4.1.3. Maintenance

Company A’s staff are annoyed that the lights switch off too often in their rooms if they work silently in front of their computer. Company A notifies the commissioning Company C about the issue and asks them to increase the delay before lights switch off.

Company C again gets the necessary authorization from the service company to interact with the BLMS. The commissioner’s tool gets the necessary authorization from BMLS to send a configuration change to all lighting devices in Company A’s offices to increase their delay before they switch off.

2.4.1.4. Decommissioning

Company A has noticed that the handheld controllers are often misplaced and hard to find when needed. So most of the time staff use the existing wall switches for manual control. Company A decides it would be better to completely remove handheld controllers and asks Company C to decommission them from the lighting system.

Company C again gets the necessary authorization from the service company to interact with the BLMS. The commissioner now deletes any rules that allowed handheld controllers authorization to control the lighting. Additionally the commissioner instructs the BLMS to push these new rules to prevent cached rules at the end devices from being used.

2.4.2. Authorization Problems Summary

- U4.1 The building owner and the companies want to be able to add new devices to their administrative domain (commissioning).
- U4.2 The building owner and the companies want to be able to integrate a device that formerly belonged to a different administrative domain to their own administrative domain (handover).
- U4.3 The building owner and the companies want to be able to remove a device from their administrative domain (decommissioning).
- U4.4 The building owner and the companies want to be able to delegate selected administration tasks for their devices to others.
- U4.5 The building owner and the companies want to be able to define context-based authorization rules.
o U4.6 The building owner and the companies want to be able to revoke granted permissions and delegations.

o U4.7 The building owner and the companies want to allow authorized entities to send data to their endpoints (default deny).

o U4.8 The building owner and the companies want to be able to authorize a device to control several devices at the same time using a multicast protocol.

o U4.9 The companies want to be able to interconnect their own subsystems with those from a different operational domain while keeping the control over the authorizations (e.g. granting and revoking permissions) for their endpoints and devices.

2.5. Smart Metering

Automated measuring of customer consumption is an established technology for electricity, water, and gas providers. Increasingly these systems also feature networking capability to allow for remote management. Such systems are in use for commercial, industrial and residential customers and require a certain level of security, in order to avoid economic loss to the providers, vulnerability of the distribution system, as well as disruption of services for the customers.

The smart metering equipment for gas and water solutions is battery driven and communication should be used sparingly due to battery consumption. Therefore the types of meters sleep most of the time, and only wake up every minute/hour to check for incoming instructions. Furthermore they wake up a few times a day (based on their configuration) to upload their measured metering data.

Different networking topologies exist for smart metering solutions. Based on environment, regulatory rules and expected cost, one or a mixture of these topologies may be deployed to collect the metering information. Drive-By metering is one of the most current solutions deployed for collection of gas and water meters.

2.5.1. Drive-by metering

A service operator offers smart metering infrastructures and related services to various utility companies. Among these is a water provider, who in turn supplies several residential complexes in a city. The smart meters are installed in the end customer’s homes to measure water consumption and thus generate billing data for the utility company. The meters do so by sending data to a base station. Several base stations are installed around the city to collect the
metering data. However in the denser urban areas, the base stations would have to be installed very close to the meters. This would require a high number of base stations and expose this more expensive equipment to manipulation or sabotage. The service operator has therefore chosen another approach, which is to drive around with a mobile base-station and let the meters connect to that in regular intervals in order to gather metering data.

2.5.2. Meshed Topology

In another deployment, the water meters are installed in a building that already has power meters installed, the latter are mains powered, and are therefore not subject to the same power saving restrictions. The water meters can therefore use the power meters as proxies, in order to achieve better connectivity. This requires the security measures on the water meters to work through intermediaries.

2.5.3. Advanced Metering Infrastructure

A utility company is updating its old utility distribution network with advanced meters and new communication systems, known as an Advanced Metering Infrastructure (AMI). AMI refers to a system that measures, collects and analyzes usage, and interacts with metering devices such as electricity meters, gas meters, heat meters, and water meters, through various communication media either on request (on-demand) or on pre-defined schedules. Based on this technology, new services make it possible for consumers to control their utility consumption and reduce costs by supporting new tariff models from utility companies, and more accurate and timely billing.

The technical solution is based on levels of data aggregation between smart meters located at the consumer premises and the Meter Data Management (MDM) system located at the utility company. Two possible intermediate levels are:

- Head-End System (HES) which is hardware and software that receives the stream of meter data and exposes an interface to the MDM.

- Data Collection (DC) units located in a local network communicating with a number of smart meters and with a backhaul interface communicating with the HES, e.g. using cellular communication.

For reasons of efficiency and cost end-to-end connectivity is not always feasible, so metering data is stored in batches in DC for some time before being forwarded to the HES, and in turn accessed by the MDM. The HES and the DC units may be operated by a third party service operator on behalf of the utility company. One
responsibility of the service operator is to make sure that meter readings are performed and delivered to the HES. An example of a Service Level Agreement between the service operator and the utility company is e.g. "at least 95% of the meters have readings recorded during the last 72 hours".

2.5.4. Authorization Problems Summary

- U5.1 Devices are installed in hostile environments where they are physically accessible by attackers. The service operator and the utility company want to make sure that an attacker cannot use a captured device to attack other parts of their infrastructure.

- U5.2 The utility company wants to restrict which entities are allowed to send data to their endpoints and to ensure the integrity of the data on their endpoints.

- U5.3 The utility company wants to control which entities are allowed to read data on their endpoints and protect such data in transfer.

- U5.4 The devices may have intermittent Internet connectivity.

- U5.5 Neither the service operator nor the utility company are always present at the time of access and cannot manually intervene in the authorization process.

- U5.6 When authorization policies are updated it is impossible, or at least very inefficient to contact all affected endpoints directly.

- U5.7 Messages between endpoints may need to be stored and forwarded over multiple nodes.

2.6. Sports and Entertainment

In the area of leisure time activities, applications can benefit from the small size and weight of constrained devices. Sensors and actuators with various functionalities can be integrated into fitness equipment, games and even clothes. Users can carry their devices around with them at all times.

Usability is especially important in this area since users will often want to spontaneously interconnect their devices with others. Therefore the configuration of access permissions must be simple and fast and not require much effort at the time of access (preferably none at all).
The required level of security will in most cases be low since security breaches will likely have less severe consequences. The continuous monitoring of data might however enable an attacker to create behavioral or movement profiles. Moreover, the aggregation of data can seriously increase the impact on the privacy of the users.

2.6.1. Dynamically Connecting Smart Sports Equipment

Jody is an enthusiastic runner. To keep track of her training progress, she has smart running shoes that measure the pressure at various points beneath her feet to count her steps, detect irregularities in her stride and help her to improve her posture and running style. On a sunny afternoon, she goes to the Finnbahn track near her home to work out. She meets her friend Lynn who shows her the smart fitness watch she bought a few days ago. The watch can measure the wearer’s pulse, show speed and distance, and keep track of the configured training program. The girls detect that the watch can be connected with Jody’s shoes and then can additionally display the information the shoes provide.

Jody asks Lynn to let her try the watch and lend it to her for the afternoon. Lynn agrees but doesn’t want Jody to access her training plan. She configures the access policies for the watch so that Jody’s shoes are allowed to access the display and measuring features but cannot read or add training data. Jody’s shoes connect to Lynn’s watch after only a press of a button because Jody already configured access rights for devices that belong to Lynn a while ago.

After an hour, Jody gives the watch back and both girls terminate the connection between their devices.

2.6.2. Authorization Problems Summary

- U6.1 Sports equipment owners want to be able to grant access rights dynamically when needed.
- U6.2 Sports equipment owners want the configuration of access rights to work with very little effort.
- U6.3 Sports equipment owners want to be able to preconfigure access policies that grant certain access permissions to endpoints with certain attributes (e.g. endpoints of a certain user) without additional configuration effort at the time of access.
- U6.4 Sports equipment owners to protect the confidentiality of their data for privacy reasons.
2.7. Industrial Control Systems

Industrial control systems (ICS) and especially supervisory control and data acquisition systems (SCADA) use a multitude of sensors and actuators in order to monitor and control industrial processes in the physical world. Example processes include manufacturing, power generation, and refining of raw materials.

Since the advent of the Stuxnet worm it has become obvious to the general public how vulnerable this kind of systems are, especially when connected to the Internet. The severity of these vulnerabilities are exacerbated by the fact that many ICS are used to control critical public infrastructure, such as power, water treatment of traffic control. Nevertheless the economical advantages of connecting such systems to the Internet can be significant if appropriate security measures are put in place.

2.7.1. Oil Platform Control

An oil platform uses an industrial control system to monitor data and control equipment. The purpose of this system is to gather and process data from a large number of sensors, and control actuators such as valves and switches to steer the oil extraction process on the platform. Raw data, alarms, reports and other information are also available to the operators, who can intervene with manual commands. Many of the sensors are connected to the controlling units by direct wire, but the operator is slowly replacing these units by wireless ones, since this makes maintenance easier.

The controlling units are connected to the Internet, to allow for remote administration, since it is expensive and inconvenient to fly in a technician to the platform.

The main interest of the operator is to ensure the integrity of control messages and sensor readings. Access in some cases needs to be restricted, e.g. the operator wants wireless actuators only to accept commands by authorized control units.

The owner of the platform also wants to collect auditing information for liability reasons.
2.7.2. Authorization Problems Summary

- U7.1 The operator of the platform wants to ensure the confidentiality of sensor data and the integrity of actuator data.
- U7.2 The operator wants to ensure that data coming from sensors and commands sent to actuators are authentic.
- U7.3 Some devices do not have direct Internet connection.
- U7.4 Some devices have wired connection while others use wireless.
- U7.5 The execution of unauthorized commands in an ICS can lead to significant financial damage, and threaten the availability of critical infrastructure services. Accordingly, the operator wants a security solution that provides a very high level of security.

3. Security Considerations

As the use cases listed in this document demonstrate, constrained devices are used in various application areas. The appeal of these devices is that they are small and inexpensive. That makes it easy to integrate them into many aspects of everyday life. Therefore, the devices will be entrusted with vast amounts of valuable data or even control functions, that need to be protected from unauthorized access. Moreover, the aggregation of data must be considered: attackers might not only collect data from a single device but from many devices, thus increasing the potential damage.

Not only the data on the constrained devices themselves is threatened, the devices might also be abused as an intrusion point to infiltrate a network. Once an attacker gained control over the device, it can be used to attack other devices as well. Due to their limited capabilities, constrained devices appear as the weakest link in the network and hence pose an attractive target for attackers.

This section summarizes the security problems highlighted by the use cases above and provides guidelines for the design of protocols for authentication and authorization in constrained RESTful environments.

3.1. Attacks

This document lists security problems that users of constrained devices want to solve. Further analysis of attack scenarios is not in scope of the document. However, there are attacks that must be considered by solution developers.
Because of the expected large number of devices and their ubiquity, constrained devices increase the danger from Pervasive Monitoring [RFC7258] attacks.

As some of the use cases indicate, constrained devices may be installed in hostile environments where they are physically accessible (see Section 2.5). Protection from physical attacks is not in the scope of ACE, but should be kept in mind by developers of authorization solutions.

Denial of service (DoS) attacks threaten the availability of services a device provides. E.g., an attacker can induce a device to perform steps of a heavy weight security protocol (e.g. Datagram Transport Layer Security (DTLS) [RFC6347]) before authentication and authorization can be verified, thus exhausting the device’s system resources. This leads to a temporary or - e.g. if the batteries are drained - permanent failure of the service. For some services of constrained devices, availability is especially important (see Section 2.3). Because of their limitations, constrained devices are especially vulnerable to denial of service attacks. Solution designers must be particularly careful to consider these limitations in every part of the protocol. This includes:

- Battery usage
- Number of message exchanges required by security measures
- Size of data that is transmitted (e.g. authentication and access control data)
- Size of code required to run the protocol
- Size of RAM memory and stack required to run the protocol

Another category of attacks that needs to be considered by solution developers is session interception and hijacking.

### 3.2. Configuration of Access Permissions

- The access control policies need to be enforced (all use cases): The information that is needed to implement the access control policies needs to be provided to the device that enforces the authorization and applied to every incoming request.

- A single resource might have different access rights for different requesting entities (all use cases).
Rationale: In some cases different types of users need different access rights, as opposed to a binary approach where the same access permissions are granted to all authenticated users.

- A device might host several resources where each resource has its own access control policy (all use cases).
- The device that makes the policy decisions should be able to evaluate context-based permissions such as location or time of access (see e.g. Section 2.2, Section 2.3, Section 2.4). Access may depend on local conditions, e.g. access to health data in an emergency. The device that makes the policy decisions should be able to take such conditions into account.

3.3. Design Considerations for Authorization Solutions

- Devices need to be enabled to enforce authorization policies without human intervention at the time of the access request (see e.g. Section 2.1, Section 2.2, Section 2.4, Section 2.5).
- Authorization solutions need to consider that constrained devices might not have internet access at the time of the access request (see e.g. Section 2.1, Section 2.3, Section 2.5, Section 2.6).
- It should be possible to update access control policies without manually re-provisioning individual devices (see e.g. Section 2.2, Section 2.3, Section 2.5, Section 2.6).

Rationale: Peers can change rapidly which makes manual re-provisioning unreasonably expensive.

- Authorization policies may be defined to apply to a large number of devices that might only have intermittent connectivity. Distributing policy updates to every device for every update might not be a feasible solution (see e.g. Section 2.5).
- It must be possible to dynamically revoke authorizations (see e.g. Section 2.4).
- The authentication and access control protocol can put undue burden on the constrained system resources of a device participating in the protocol. An authorization solutions must take the limitations of the constrained devices into account (all use cases, see also Section 3.1).
- Secure default settings are needed for the initial state of the authentication and authorization protocols (all use cases).
Rationale: Many attacks exploit insecure default settings, and experience shows that default settings are frequently left unchanged by the end users.

- Access to resources on other devices should only be permitted if a rule exists that explicitly allows this access (default deny) (see e.g. Section 2.4).

- Usability is important for all use cases. The configuration of authorization policies as well as the gaining access to devices must be simple for the users of the devices. Special care needs to be taken for home scenarios where access control policies have to be configured by users that are typically not trained in security (see Section 2.2, Section 2.3, Section 2.6).

3.4. Proxies

In some cases, the traffic between endpoints might go through intermediary nodes (e.g. proxies, gateways). This might affect the function or the security model of authentication and access control protocols e.g. end-to-end security between endpoints with DTLS might not be possible (see Section 2.5).

4. Privacy Considerations

Many of the devices that are in focus of this document register data from the physical world (sensors) or affect processes in the physical world (actuators), which may involve data or processes belonging to individuals. To make matters worse the sensor data may be recorded continuously thus allowing to gather significant information about an individual subject through the sensor readings. Therefore privacy protection is especially important, and Authentication and Access control are important tools for this, since they make it possible to control who gets access to private data.

Privacy protection can also be weighted in when evaluating the need for end-to-end confidentiality, since otherwise intermediary nodes will learn the content of potentially sensitive messages sent between endpoints and thereby threaten the privacy of the individual that may be subject of this data.

In some cases, even the possession of a certain type of device can be confidential, e.g. individuals might not want to others to know that they are wearing a certain medical device (see Section 2.3).

The personal health monitoring use case (see Section 2.3) indicates the need for secure audit logs which impose specific requirements on a solution. Auditing is not in the scope of ACE. However, if an
authorization solution provides means for audit logs, it must consider the impact of logged data for the privacy of all parties involved. Suitable measures for protecting and purging the logs must be taken during operation, maintenance and decommissioning of the device.

5. Acknowledgments

The authors would like to thank Olaf Bergmann, Sumit Singhal, John Mattson, Mohit Sethi, Carsten Bormann, Martin Murillo, Corinna Schmitt, Hannes Tschofenig, Erik Wahlstroem, and Andreas Backman for reviewing and/or contributing to the document. Also, thanks to Markus Becker, Thomas Poetsch and Koojana Kuladinithi for their input on the container monitoring use case.

Ludwig Seitz and Goeran Selander worked on this document as part of EIT-ICT Labs activity PST-14056.

6. IANA Considerations

This document has no IANA actions.

7. Informative References

[Jedermann14]


Authors’ Addresses
Abstract

Authentication and authorization are fundamental security features used in Internet and Web applications. Providing the same level of security functionality to the Internet of Things (IoT) environment as well is a logical enhancement and reduces the risk of unauthorized access to personal data.

IoT devices, however, have limitations in terms of processing power, memory, user interface, Internet connectivity, etc. Since many use cases span Web and IoT environments and the question of "Web" vs. "IoT" can in some cases be considered a continuum, it is required to find security solutions that can accommodate the capabilities and constraints of both environments without significant compromises.

Thus, an approach of adapting already standardized and deployed authentication and authorization technologies is worth examining. This document describes how the Web Authorization Protocol (OAuth) in combination with User-Managed Access (UMA) can be used for an IoT environment to bring Web-scale authorization services to the IoT world.

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 September 10, 2015.

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. Use Cases .................................................... 3
   3.1. Using OAuth with Scales ................................. 4
   3.2. Using UMA with Scales ................................. 6
   3.3. Using OAuth and UMA with Cars .................... 8
   3.4. Using OAuth and UMA with Door Locks ............. 9
4. Protocol Designs for the Web and Beyond .................. 10
5. Instantiations ............................................... 11
   5.1. Car Use Case ......................................... 12
   5.2. Door Lock Use Case ................................ 14
6. UMA Use Case Mapping Exercise ........................... 16
7. Security Considerations .................................... 18
8. IANA Considerations ........................................ 19
9. Acknowledgements ........................................... 19
10. References .................................................. 19
   10.1. Normative References ............................... 19
   10.2. Informative References ............................ 21
Authors’ Addresses ............................................. 21

1. Introduction

Deciding when a certain use case falls under the category of IoT and when it is not turns out to be a difficult task. For this reason, [RFC7228] made an attempt to describe characteristics of constrained-node networks and highlights some of the challenges. Companies often
have some degree of freedom to make trade-off decisions, for example, in terms of cost vs. physically available resources to push the boundaries of what can be done with IoT devices.

Manufacturers must take not only hardware costs into account, but also software development costs; reusing existing software, standards, practices, and expertise can help to lower the total cost of a product. Hence, the use cases combine the already existing identity and access management infrastructure with access control to objects in the physical world.

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document leverages terminology from [RFC6749] and [I-D.hardjono-oauth-umacore]. Especially pertinent definitions are paraphrased below.

Resource Owner: An entity capable of granting access to a protected resource.

Resource Server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Authorization Server: The server issuing access tokens to the client after successfully authorizing it.

Requesting Party: An entity (which may or may not be the same as the resource owner) that uses a client to seek access to a protected resource.

Client: An application making protected resource requests with the resource owner’s authorization and on the requesting party’s behalf.

3. Use Cases

The sub-sections below illustrate some use cases that start with classic OAuth functionality and then extend it to functionality only available with UMA-based environments. The scenarios involve Web, smart phone app, and IoT devices. Unlike the scenarios described in [I-D.ietf-ace-usecases] this write-up is not solution agnostic but
instead aims to take the OAuth/UMA solutions into account. In a stepwise refinement we then add even more details in Section 5.

3.1. Using OAuth with Scales

In a classic OAuth flow, an end-user (the resource owner) can enable a client application to call an API (at the resource server) on his or her behalf securely and with authorized consent, without having to reveal his or her credentials, such as a username and password, to the client. An app-specific access token (issued by the authorization server at which the resource owner is able to authenticate), whose operation may be scoped to some subset of the API’s capabilities, is substituted for the long-term credentials instead.

The basic OAuth architecture is shown in Figure 1 and the corresponding message exchange in Figure 2.

Figure 1: OAuth Architecture.
We can apply a similar pattern to IoT devices as well. For example, envision an end-user Alice and her new purchase of an Internet-connected scale designed for "quantified self" scenarios. In our example, the scale has a micro-controller that was pre-provisioned with a certificate during manufacturing enabling the device to authenticate itself to the vendor-authorized software update server as well as to other parties. The identifier used for authentication of a scale is something as benign as an EUI-64 serial number.

Once the identifier used by the scale and Alice’s account information have been provisioned into an online repository, and if Alice can demonstrate appropriate control of the device -- for example, by entering a confirmable PIN code or serial number that was packaged with the shipped device into her online account record, whether through a Web or mobile app -- it is possible to treat the device as an OAuth client and issue it an OAuth token so that it can act on Alice’s behalf.

The value of this association is that any API calls made by the scale, for example to report Alice’s weight, body mass index (BMI), or progress against health goals into her online account, will be associated with her alone. If other household members use the scale as well, their unique associations will ensure that their data will go to the right place (assuming there is a mechanism at the scale
that allows family members to be differentiated). Further, each token can be revoked and expired exactly like any other OAuth token.

3.2. Using UMA with Scales

UMA builds on top of OAuth (and optionally OpenID Connect [OIDC]) to let an end-user achieve three main goals:

1. authorize other parties to access APIs under his or her control using client applications;

2. set conditions for access so that those other parties may have to provide "claims" and do step-up authentication to get access (in a so-called claims gathering process); and

3. centralize management of all these conditions for access in one cloud service.

The basic architecture and flow is shown in Figure 3. A protection API token (PAT) is an OAuth token with a scope that gives the resource server access to the UMA-standardized protection API at the authorization server; an authorization API token (AAT) is an OAuth token with a scope that gives the client access to the UMA-standardized authorization API; and a requesting party token (RPT) is the main access token issued to a requesting party, which does not rely on resource owner presence for issuance.
UMA can be thought of as "OAuth++", in that it adds two major elements: a formal protection API presented by the authorization server, so that resource servers running in different domains can be "authorization relying parties" to it, and the "requesting party" concept distinct from the resource owner (as discussed in Section 2).

The requesting party may be required to interact with the authorization server when the client asks for permission to access a resource. However, if this interaction requires authentication, this authentication step may be outsourced to a variety of different identity providers, including the client (which may be allowed to "push" identity claims to the authorization server), the authorization server itself, or any other identity provider, with the authorization server functioning as a relying party in this case.

Similarly to the previous use case in Section 3.1, there is value in extending the Web world to the world of devices because the data...
originating in a device often travels to the cloud. Alice may want to share her scale data with friends, with her doctor, or in anonymized form with a public health service.

The benefit of using an UMA authorization server, requesting party tokens, and so on to manage Alice’s control of her doctor’s and others’ access to the data her scale generates is that she:

1. does not have to be present when they request access, crafting policies prior to access attempts or handling access approval requests after attempts;

2. can demand that requesting parties present proof of their suitability (such as current valid hospital credentials);

3. can change the length permission validity, including revoking sharing relationships;

4. can set policies governing clients used by requesting parties as well; and

5. can do this from a centralizable authorization point, crossing multiple resource servers (and thus devices feeding into them).

3.3. Using OAuth and UMA with Cars

A connected car example illustrates other desirable aspects of IoT authentication and authorization.

Alice buys a new car. At manufacture time, the car was registered at the manufacturer’s authorization server. When buying the car, Alice can create an account at the manufacturer’s website and reuse the already configured authorization server. Alice installs a car managing mobile app on her phone to manage her car. Alice authorizes the app to act on her behalf as OAuth client to perform actions, such as open car door, which would be similar to authorizing an app to send tweets on my behalf to the twitter API but in this case the resource server is the car and the API is accessed over Bluetooth Smart.

Since the operation of opening the car is security sensitive, it is desirable to require more than a long term access token to open the door and to start the car. So instead of just accepting the access token the authorization server may require Alice to supply more information and a UMA claims gathering process is started, such as requiring a multi-factor authentication using a fingerprint or a PIN code on her phone.
Furthermore, Alice wants to share driving rights with her husband Ted. Alice is owner of the car and is authorized to add new drivers to the car. To do this Alice can setup the policies at the authorization service governing who can do what with the car at what time. Alice configures a rule that allows Ted to request a token for the scope of driving the car, but just as Alice, Ted is required to download the app, authorize it and go through a claims gathering flow to actually get the token to start the car using his smart phone app.

With this delegation of rights to the car Ted could potentially even create a valet key with geo fenced driving range and no access to trunk when he leaves the car in a parking garage and thereby create a valet key for the physical world.

The use of standardized protocols allows Alice to use her own authorization server. Alice could choose to unregister the car at the manufacturer authorization server and register the car to an authorization server of her liking. The car would register available resources and scopes and Alice could configure policies as above using her own authorization server.

Since cars are not always located in areas with Internet connectivity it is envisioned that cars need to be able to verify access tokens locally (without the need to consult an authorization server in real-time). Once the car is online again it could check whether any new revocation information is available and upload information about earlier authorization decisions to the audit log.

A similar situation may occur when Alice asks her friend Trudy to get the groceries from the trunk of her car (which she forgot there earlier) while they are at their remote summer cottage. Without Internet connectivity Alice cannot delegate access to her car to Trudy using the authorization server located in the cloud. Instead, she transfers an access token to Trudy using Bluetooth. This access token entitles Trudy to open the trunk but not to drive it and grants those permissions only for a limited period. To ensure that the car can actually verify the content of the access token the client app of Alice again uses the capabilities of the proof-of-possession tokens.

3.4. Using OAuth and UMA with Door Locks

Alice, the owner of a small enterprise, buys a door lock system for her office. She would expect to be able to provision policies for access herself, in effect acting as "system administrator" for herself and for her five employees. She may also want to choose her own authorization server, since she wants to integrate the physical access control system with the rest of the resources in her company and the enterprise identity management system she already owns. She
wants to control the cloud-based file system, financial and health data, as well as the version control and issue tracking software.

4. Protocol Designs for the Web and Beyond

The design of OAuth was intentionally kept flexible to accommodate different deployment situations. For example, authentication of the resource owner to the authorization server before granting access is not standardized and different authentication technologies can be used for that purpose. The user interface shown to the resource owner when asking for access to the protected resource is not standardized either.

Over the years various extensions have been standardized to the core OAuth protocol to reduce the need for proprietary extensions that offer token revocation, an access token format called JSON Web Token, or proof-of-possession tokens that offer an alternative security model for bearer tokens [RFC6750].

Due to the nature of the Web, OAuth protocol interactions have used HTTPS as a transport; however, other transports have been investigated as well, such as OAuth for use over SASL (for use with email) and more recently OAuth over the Constrained Application Protocol (CoAP).

This document provides the reader with information about which OAuth extensions will be useful for the IoT context. In its structure it is very similar to the DTLS/TLS IoT profile document that explains what TLS extensions and ciphersuites to use for different IoT deployment environments. Interestingly, very little standardization effort is necessary to make OAuth and UMA fit for IoT. To a large extend the work is centered around using alternative transports (such as CoAP and DTLS instead of HTTP over TLS) to minimize the on-the-wire overhead and to lower code-size and to define profiles for highly demanded use cases.

The UMA group, benefiting from observing the OAuth experience and from the era in which UMA itself has been developed, has built extension points into the protocol, already anticipating a need for flexibility in transport bindings. Thus, UMA has three "extensibility profiles" that enable alternate bindings (such as CoAP) to be defined for communications between an authorization server and resource server, a resource server and client, and an authorization server and client respectively. It also, similarly to OAuth, as other extensibility options, such as token profiling and the ability to extend JSON formats to suit a variety of deployment needs.
5. Instantiations

In this section we provide additional details about the use of OAuth and UMA for solving the use cases outlined in Section 3. In general, the following specifications are utilized:

- **OAuth 2.0** [RFC6749] for interacting with the authorization server. The use of the CoAP-OAuth profile [I-D.tschofenig-ace-oauth-iot] maybe used but is not essential for the examples in this section since the client is less constrained.

- **Bearer tokens and proof-of-possession tokens** as two different security models for obtaining and presenting access tokens. Bearer tokens are defined in [RFC6750] and the architecture for proof-of-possession (PoP) tokens can be found at [I-D.ietf-oauth-pop-architecture]. PoP tokens introduce the ability to bind credentials, such as an ephemeral public key, to the access token.

- **UMA** [I-D.hardjono-oauth-umacore] for registering the resource server with the authorization server provided by Alice and for management of policy.

- **Dynamic Client Registration** [I-D.ietf-oauth-dyn-reg] for the client app to register at the authorization server.

- **Token introspection** [I-D.ietf-oauth-introspection] for optionally allowing the resource server to verify the validity of the access token (if this step is not done locally at the resource server). The use of token introspection over CoAP [I-D.wahlstroem-ace-oauth-introspection] reduces overhead.

- **JSON Web Token (JWT)** [I-D.ietf-oauth-json-web-token] for the format of the access token. JSON Web Signatures (JWT) [I-D.ietf-jose-json-web-signature] are used for creating a signature over the JWT. The use of a CBOR encoding of various JSON-based security specifications is under discussion to reduce the size of JSON-based tokens.

- A new Bluetooth Smart service and profile for conveying access tokens securely from the client to the resource server. If CoAP runs between the client and a constrained resource server then [I-D.tschofenig-ace-oauth-bt] provides additional overhead reduction.
5.1. Car Use Case

In the car use case, as described in Section 3.3, the car acts as the resource server and an application on the smart phone plays the role of the client. Alice is first a delegated administrator then becomes a resource owner of the car.

Alice creates an account, downloads and authorizes the mobile app:

1. Alice creates an account on manufacturer’s website.

2. Alice selects that two factor authentication must be used to be able to start controlling car from an app.

3. Alice downloads app and starts it.

4. App has never been provisioned so a browser is started, user selects manufacturer’s authorization server from a list.

5. Alice authenticates using two factors and authorizes the application.

6. Access and refresh tokens are provisioned to the app.

Alice configures policies to add Tim as new driver:

1. Alice opens the car-settings page within the app.

2. Alice selects to add a new driver by supplying Tims email address.

3. Alice checks the checkboxes that also makes Tim a delegated administrator.

4. Alice saves the new policies.

Alice opens car door over Bluetooth Smart:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants to open the car door.

2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the car Alice is seeking access to.

3. The authorization server evaluates the request to open the car door on the specific car and verifies it against the access
control policy. Note that the app authenticated itself to the authorization server.

4. The authorization server prompts Alice for a PIN code using claims gathering.

5. Alice enters pin and the application communicates it to the authorization server.

6. It turns out that the system administrator has granted her access to that specific car and she is given access by returning an access token.

7. The smart phone app then uses the obtained access token to create a request (which includes the access token) over Bluetooth Smart using on the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

8. The car receives the request and verifies it.

9. To check whether the permissions are still valid the car sends the access token to the introspection endpoint.

10. The authorization server validates the access token and returns information about the validity of the token to the car. In this case it’s a valid token.

11. The request is logged.

12. The car gets a response and opens the car door.

Alice changes authorization server:

1. Alice wants to connect the car to her own authorization server instead of the manufacturers default authorization server.

2. Alice makes a request to the current authorization server to unbind the device from the authorization server.

3. The authorization server validates Alice request to remove the authorization server.

4. Alice configures a new authorization server in the apps UI.
5. The app starts an authorization code grant flow with the private authorization server of Alice. Alice logs on and authorizes the app to act on her behalf.

6. The app sends information about the new authorization server to the car using Bluetooth Smart.

7. The car registers the resource it offers with the new authorization server.

8. Alice configures herself as the car owner in the new authorization server.

9. The car unbinds itself from the old authorization server by invalidating the access tokens using the revocation endpoint.

5.2. Door Lock Use Case

In the constrained server use case, as described in Section 3.4, the door lock acts as the resource server and an application on the smartphone plays the role of the client.

Since the client runs on a powerful smartphone standard OAuth according to OAuth Core can be used. To avoid leakage of the access token the use of a proof-of-possession token is utilized instead of a bearer token. This allows the client to demonstrate the possession of the private key to the client. Both symmetric as well as asymmetric cryptography can be used. The use of asymmetric cryptography is beneficial since it allows the client to create a public / private key pair and to never expose the private key to other parties.

As a setup-step the following steps are taken as part of the enterprise IT

1. Alice, as the enterprise network administrator and company owner, enables the physical access control rights at the identity management server.

2. Alice downloads the enterprise physical access control system app on her phone. By downloading the app she agrees to the terms of use and she accepts the permissions being asked for by the app.

3. Alice associates her smart phone app with her account by login into the enterprise management software, which uses OAuth 2.0 for delegating access to the app.
4. Alice, as the enterprise administrator, configures policies at the authorization server to give her employees access to the office building as well.

5. In this use case each door lock is provisioned with an asymmetric key pair and the public key of the authorization server. The public key of each door lock is registered with the authorization server. Door locks use these keys when interacting with the authorization server (for authentication in case of token introspection), for authenticating towards the client, and for verifying the signature computed over the access token.

When Alice uses her smartphone for the first time to access the office building the following steps take place:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants access.

2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the door Alice is seeking access to. The request is protected using TLS.

3. The authorization server evaluates the request and verifies it against the access control policy. Since Alice has added herself to access control policies already she is given access by returning an access token. This access token includes the fingerprint of the public key provided in the request. The access token is digitally signed to avoid any modification of the content.

4. The smart phone app then uses the obtained information to create a request (which includes the access token) over Bluetooth Smart using the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

5. The door lock software receives the request and verifies the digital signature, inspects the content (such as expiry date, and scope), and determines whether the fingerprint of the public key corresponds to the private key used by the client. Once successfully verified the door is unlocked, and Alice is allowed to enter.

6. The physical access control app caches the access token for future use.
As a variation of the above-described procedure, the door lock might consult the authorization server using token introspection to determine the validity of the access token. This allows the enterprise system software to make real-time access control decisions and to better gain visibility about the number of employees in the building (in case of an emergency).

When Alice approaches the door next time her physical access control app determines that a cached (and still valid) access token is available and no further interaction with the authorization server is needed. Decisions about how long to cache access tokens are a policy decision configurable into the system and impact the performance of the protocol execution.

When Bob, who is employed by Alice, approaches the office building for the first time his downloaded physical access control app also interacts with the door. While Bob still has to consent to the use of app, Alice does not need to authorize access of Bob to the office building in real-time since she has already granted access to her employees earlier already.

6. UMA Use Case Mapping Exercise

An analysis of [I-D.hardjono-oauth-umacore] suggests that its capabilities have a good architectural match with many published ACE use cases. The following are aggregated and paraphrased versions of use cases discussed in [I-D.ietf-ace-usecases]:

Owner grants different resource access rights to different parties (U1.1, U2.3, U.3.2):

UMA meets this use case because the requesting party is formally distinct from the resource owner and because each requesting party, and each client, is represented distinctly at each authorization server, able to have differential policy applied to it.

Owner grants different access rights for different resources on a device (U1.3, U4.4, U5.2):

UMA meets this use case because the resource server is able to register each resource set (according to boundaries it unilaterally determines) at the authorization server, so that the resource owner can apply policy to it distinctly.

Owner not always present at time of access (U1.6, U5.5):
UMA meets this use case because it is a profile of OAuth that defines an asynchronous authorization grant, meaning that the client’s interactions during a resource access attempt do not require a resource owner’s interaction.

Owner grants temporary access permissions to a party (U1.7):

UMA meets this use case because the default, mandatory-to-implement permissions associated with a requesting party token (the "bearer" profile) are able to be time-limited and are in a time-limatable JSON Web Token as well.

Owner applies verifiable context-based conditions to authorizations (U2.4, U4.5, U6.3):

UMA meets this use case because a resource owner can configure an authorization server with policies, or an authorization server can apply system-default policies, to demand "trust elevation" when a client requests authorization data, such that a requesting party or client must satisfy authentication, claims-based, or (through extension) any other criteria prior to being issued authorization data.

Owner preconfigures access rights to specific data (U3.1, U6.3):

UMA meets this use case because it defines an asynchronous authorization grant, as described above. Preconfiguration is a case when a resource owner sets policy prior to an access attempt.

Owner adds a new device under protection (U4.1):

UMA meets this use case because it enables a resource owner to associate a device and its corresponding resource server with an authorization server through consenting to the issuance of a protection API token (PAT), enabling the resource server to outsource protection of its resources to the authorization server.

Owner puts a previously owned device under protection (U4.2):

UMA meets this use case because a previous resource owner can revoke a pre-existing PAT if one existed, revoking the previous consent in place, and the new owner can mint a new PAT.

Owner removes a device from protection (U4.3):

UMA meets this use case because the resource owner can revoke the PAT.
Owner revokes permissions (U4.6):

UMA meets this use case because the resource owner can configure the authorization server to revoke or terminate an existing permission. The default, mandatory-to-implement requesting party token profile ("bearer") requires runtime token introspection, ensuring relatively timely retrieval of a revoked permission (barring authorization server caching policy). Other profiles may have different results.

Owner grants access only to authentic, authorized clients (U7.1, U7.2):

UMA meets this use case because it enables OAuth as well as OpenID Connect authentication of clients, including dynamic authentication, and also enables resource owners to configure authorization servers with policy, such that only desired clients wielded by desired requesting parties are given access to the owner’s resources.

7. Security Considerations

This specification re-uses several existing specifications, including OAuth and UMA, and hence the security-related discussion in those documents is applicable to this specification. A reader is encouraged to consult [RFC6819] for a discussion of security threats in OAuth and ways to mitigate them. On a high level, the security guidance provided in [I-D.iab-smart-object-architecture] will help to improve security of Internet of Things devices in general.

Despite all the available guidance it is nevertheless worthwhile to repeat the most important aspects regarding the use of access tokens, which are a core security mechanism in the OAuth / UMA specifications.

Safeguard bearer tokens: Client implementations MUST ensure that bearer tokens are not leaked to unintended parties, as they will be able to use them to gain access to protected resources. This is the primary security consideration when using bearer tokens and underlies all the more specific recommendations that follow. This document also outlines the use of proof-of-possessions, which provide stronger security properties than bearer tokens and their use is RECOMMENDED.

Validate TLS certificates: TLS/DTLS clients MUST validate the certificates received during the handshaking procedure. TLS/DTLS is used heavily in OAuth/UMA between various parties. Failure to verify certificates will enable man-in-the-middle attacks.
Always use TLS/DTLS: The use of TLS/DTLS is mandatory for use with OAuth as a default. Particularly when bearer tokens are exchanged the communication interaction MUST experience communication security protection using TLS (or DTLS). Failing to do so exposes bearer tokens to third parties and could consequently give attackers unintended access. Proof-of-possession tokens on the other hand do not necessarily require the use of TLS/DTLS but TLS/DTLS is RECOMMENDED even in those cases since TLS/DTLS offers many desirable security properties, such as authentication of the server side.

Issue short-lived tokens: Authorization servers SHOULD issue short-lived tokens. Using short-lived bearer tokens reduces the impact of them being leaked and allows easier revocation in scenarios where resource servers are offline.

Issue scoped tokens: Authorization servers MUST issue tokens that restrict tokens for use with a specific resource server and contains appropriate entitlements to control access in a fine-grained fashion.

8. IANA Considerations

This document does not require actions by IANA.

9. Acknowledgements

This is the first version of the document. We appreciate feedback.

10. References

10.1. Normative References

[I-D.hardjono-oauth-umacore]

[I-D.ietf-jose-json-web-signature]

[I-D.ietf-oauth-dyn-reg]
ietf@justin.richer.org, i., Jones, M., Bradley, J.,
Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client
Registration Protocol", draft-ietf-oauth-dyn-reg-24 (work
in progress), February 2015.

[I-D.ietf-oauth-introspection]
ietf@justin.richer.org, i., "OAuth 2.0 Token
Introspection", draft-ietf-oauth-introspection-05 (work in
progress), February 2015.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
(JWT)", draft-ietf-oauth-json-web-token-32 (work in
progress), December 2014.

[I-D.ietf-oauth-pop-architecture]
Hunt, P., ietf@justin.richer.org, i., Mills, W., Mishra,
P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession
(PoP) Security Architecture", draft-ietf-oauth-pop-
architecture-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-bt]
Tschofenig, H., "The OAuth 2.0 Bearer Token Usage over the
Constrained Application Protocol (CoAP)", draft-
tschofenig-ace-oauth-bt-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-iot]
Tschofenig, H., "The OAuth 2.0 Internet of Things (IoT)
Client Credentials Grant", draft-tschofenig-ace-oauth-
iot-01 (work in progress), March 2015.

[I-D.wahlstroem-ace-oauth-introspection]
Wahlstroem, E., "OAuth 2.0 Introspection over the
Constrained Application Protocol (CoAP)", draft-
wahlstroem-ace-oauth-introspection-00 (work in progress),
October 2014.

[OIDC]
Sakimura, N., "OpenID Connect Core 1.0 incorporating
Errata Set 1", http://openid.net/specs/openid-connect-core-1_0.html,
November 2014.

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate

[RFC6749]
6749, October 2012.
10.2. Informative References

[I-D.iab-smart-object-architecture]

[I-D.ietf-ace-usecases]


Authors’ Addresses

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Eve Maler
Forgerock

Email: eve.maler@forgerock.com

Erik Wahlstroem
Nexus Technology
Sweden

Email: erik.wahlstrom@nexusgroup.com
URI: https://www.nexusgroup.com
Samuel Erdtman
Nexus Technology
Sweden

Email: samuel.erdtman@nexusgroup.com
URI: https://www.nexusgroup.com
Problem Description for Authorization in Constrained Environments
draft-seitz-ace-problem-description-03

Abstract

We present a problem description for authentication and authorization in constrained-node networks, i.e. networks where some devices have severe constraints on memory, processing, power and communication bandwidth.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License 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
1. Introduction

Authorization is the process of deciding what an entity ought to be allowed to do. This memo is about properties of security protocols to enable explicit and dynamic authorization of clients to access a resource at a server, in particular in constrained environments when the client and/or server are constrained nodes.

Relevant use cases are provided in [I-D.ietf-ace-usecases], which also lists some authorization problems derived from the use cases. In this memo we present a more specific problem description for authentication and authorization in constrained RESTful environments together with a detailed set of assumptions and requirements (cf. section 4).

1.1 Terminology

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

RESTful terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

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

"Explicit" authorization is used here to describe the ability to specify in some detail which entity has access to what and under what conditions, as opposed to "implicit" authorization where an entity is either allowed to access everything or nothing.

"Dynamic" authorization means that the access control policies and the parameters on which they are evaluated may change during normal operations, as opposed to "static" authorization meaning that access control policies cannot be changed during normal operations and may require some special procedure such as out-of-band provision.

2. Background

We assume a client-server setting, where a client wishes to access some resource hosted by a server. Such resources may e.g. be sensor data, configuration data, or actuator settings. Thus access to a resource could be by different methods, some of which change the state of the resource. In this memo, we consider the REST setting
i.e. GET, POST, PUT and DELETE, and application protocols in scope are HTTP [RFC7231] and CoAP [RFC7252].

We assume that the roles of client and server are not fixed, i.e. a node which is client could very well be server in some other context and vice-versa. Further we assume that in some cases, clients are not previously known to servers, thus we cannot assume that the server has access control policies specific to that client when the client initiates communication.

Finally we also assume that in a significant number of cases, the server and/or the client are too constrained to handle the evaluation of complex access control policies and related configuration on their own. Many authorization solutions involve a centralized, trusted third party, supporting the client and/or resource server. A trusted third party provides a more scalable way to centrally manage authorization policies, in order to ensure consistent authorization decisions. The physical separation of policy decision and policy enforcement is an established principle in policy based management, e.g. [RFC2748].

Borrowing from OAuth 2.0 [RFC6749] terminology we name the entities: client (C), resource server (RS), authorization server (AS - the third party), and resource owner (RO). RO is in charge of the access control policies implemented in the AS governing the actions of RS. However, the RO need not be active in a constrained device access control setting, so we cannot rely on timely interactions with the RO. In the target setting RS is typically constrained, C may be constrained, whereas AS is not assumed to be constrained.

Since RS is constrained, we assume that it needs to offload authorization policy management and/or authorization decision making to AS. This means that some authorization information needs to be transferred from AS to RS.

Protecting information carried between AS and RS, requires some a priori established cryptographic keys. How those keys are established is out of scope for this problem description.

AS may for example be implemented as a cloud service, in a home server, or in a smartphone. C and RS may or may not have connectivity to AS at the time of the access request, e.g. because they cannot handle multiple, simultaneous connections. Another reason for intermittent connectivity may be that constant connectivity is not affordable (e.g. due to limited battery power, or a sensor mobility business case for which cellular connectivity cost too much or is not available). Obviously, in order for a client request to reach RS there must be connectivity between C and RS, but
that could be a short range technology such as Bluetooth, ZigBee, or NFC. Furthermore, if there is not sufficient authorization information about C in RS, and neither C nor RS can access AS, access requests will be denied. Therefore we assume that either C or RS can access AS at some point in time, prior to the client’s request.

As a summary, there are potentially three information flows that needs to be protected (see Figure):

1. The transfer of authorization information from AS to RS
2. The transfer of cryptographic keys or credentials from AS to RS and C, respectively
3. The access request/response procedure between C and RS

Figure. Information flows that needs to be protected.
Only showing origin and destination, actual flow may pass intermediary nodes.

NOTE:

The information flow in 1. above enables RO to control the interactions of a constrained RS by means of access control policies. There is an ongoing discussion about an analogous information flow enabling the stakeholder associated to C ("Requesting Party" in UMA

Seitz & Selander
Expires September 10, 2015
terminology [I-D.hardjono-oauth-umacore]) to control the interactions of a constrained C by means of policies. While this would not be policies for access control to resources, it could be useful in certain settings which require dynamically changing interaction patterns with a constrained client without updating firmware. Such a solution could potentially reuse all security components required to protect the information flow in 1., so no additional specifications would be needed. This aspect is not discussed further in this draft.

3. Problem Description

A number of problems needs to be solved in order to achieve explicit and dynamic authorization, as is described in this section.

3.1. Authorization

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

- AS needs to transfer authorization information to RS.
- 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.
- The RS needs to be able to verify the authenticity of the authorization information. There is a trade-off here between processing complexity and deployment complexity.
- The RS needs to enforce the authorization decisions of the AS. The authorization information it obtained from AS might require additional policy evaluation (e.g. matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the RS needs to be adapted to the capabilities of the constrained device.
- 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 RS at different times and in different ways prior to or during the client request.

3.2. Authentication

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

- RS need to authenticate AS to ensure that the authorization information and related data comes from the correct source.
- C may need to authenticate AS to ensure that it gets security information related to the resources from the right source.
- In some use cases RS needs to authenticate some property of C, in order to bind it to the relevant authorization information. In other use cases, authentication and authorization of C may be implicit, e.g. by encrypting the resource representation the RS only providing access to those who possess the key to decrypt.
- C may need to authenticate RS, in order to ensure that it is interacting with the right resources. Alternatively C may just verify the integrity of a received resource representation.
- AS may need to authenticate its communication partner (either C or RS), in order to ensure it serves the correct device.

3.3. 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:

- 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.

- An alternative is object security at application layer, e.g. using [I-D.selander-ace-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.koster-core-coapmq]). A problem with object security is that it can not provide confidentiality for the message headers.

- 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.

3.4. 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. Do we want to support solutions based on asymmetric keys or 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 e.g. in terms of deployment.

- **Key Establishment**
  
  How are the corresponding cryptographic keys established? Considering section 3.1 there must be a binding 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 [I-D.ietf-ace-usecases] describes spontaneous change of access policies - e.g. 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?

4. 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.

4.1 Architecture
The architecture consists of at least the following types of nodes:

- RS hosting resources, and responding to access requests
- C requesting access to resources
- AS supporting the access request/response procedure by providing authorization information to RS.
  - AS may also provide other services such as authenticating C on behalf of RS, or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- The architecture may contain intermediary nodes between any pair of C, RS and AS, such as e.g. forward/reverse proxies in the CoRE architecture. The solution shall not unduly restrict the use of intermediaries.
  - The architecture shall support session based security and data object security.

4.2 Constrained Devices

- 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
- AS is not a constrained device.

- All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.
  - We assume the use of a standardized symmetric key algorithm, such as AES.
- Except for the most constrained devices we assume the use of a standardized cryptographic hash function such as SHA-256.

- Public key cryptography requires additional resources (e.g. RAM, ROM, power, specialized hardware).

- A DTLS handshake involves significant computation, communication, and memory overheads in the context of constrained devices.

- The RAM requirements of DTLS handshakes with public key cryptography are prohibitive for certain constrained devices.

- Certificate-based DTLS handshakes require significant volumes of communication, RAM (message buffers) and computation.

- The solution shall support a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. section 5.2).

4.3 Authentication

- RS need to authenticate AS to ensure that the authorization information and related data comes from the correct source.

- Depending on use case, C, RS or AS may need to authenticate each other.

4.4 Authorization

- The authorization decision is based on credentials presented by C, the requested resource, the RESTful method, and local context in RS at the time of the request, or on any subset of this information.

- The authorization decision is taken either by AS or RS.

- The authorization decision is enforced by RS.

- RS needs to have access to 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.

- Apart from authorization for access to a resource, authorization may also be required for access to information about a resource.
(e.g. resource descriptions).

- The solution may need to be able to support the delegation of access rights.

4.5 Authorization Information

- Authorization information is transferred from AS to RS using Agent, Push or Pull mechanisms [RFC2904].
- RS shall authenticate that the authorization information is coming from AS.
- The authorization information may also be encrypted end-to-end between AS and RS.
- RS may not be able to communicate with AS at the time of the request from C.
- RS may store or cache authorization information.
- Authorization information may be pre-configured in RS.
- Authorization information stored or cached in RS shall be possible to change. The change of such information shall be subject to authorization.
- Authorization policies stored on RS may be handled as a resource, i.e. information located at a particular URI, accessed with RESTful methods, and the access being subject to the same authorization mechanics. AS may have special privileges when requesting access to the authorization policy resources on RS.
- There may be mechanisms for C to look up the AS which provides authorization information about a particular resource.

4.6 Resource Access

- Resources are accessed in a RESTful manner using GET, PUT, POST, DELETE.
- By default, the resource request shall be integrity protected and may be encrypted end-to-end from C to RS. It shall be possible for RS to detect a replayed request.
- By default, the response to a request shall be integrity protected and encrypted end-to-end from RS to C. It shall be possible for C to detect a replayed response.
o RS shall be able to verify that the request comes from an authorized client.

o C shall 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).

4.7 Keys and Cipher Suites

o AS and RS have established cryptographic keys. Either AS and RS 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 can be 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.

o There must be a mechanism for C to establish the necessary key(s) to verify and decrypt the response.

o There must be a mechanism for C to look up the supported cipher suites of a RS.

4.8 Network Considerations

o The solution shall prevent network overload due to avoidable communication with AS.

o The solution shall prevent network overload by compact authorization information representation.

o The solution shall optimize the case where authorization information does not change often.

o The solution where possible shall support an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

4.9 Legacy Considerations
The solution shall work with existing infrastructure.

The solution shall support authorization of access to legacy devices.

5. Security Considerations

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments are provided in e.g. OAuth 2.0 [RFC6749].

In this section we focus on specific security aspects related to authorization in constrained-node networks.

5.1 Physical Attacks on Sensor and Actuator Networks

The focus of this work is on constrained-node networks consisting of connected 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:

- Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.
- 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. For example:

- Instead of eavesdropping the sensor data or attacking the authorization system to gain access to the data, the attacker could make its own measurements on the physical object.
- Instead of manipulating the sensor data the attacker could change the physical object which the sensor is measuring, thereby changing the payload data which is being sent.
- Instead of manipulating data for an actuator or attacking the authorization system, the attacker could perform an unauthorized action directly on the physical object.
o A denial-of-service attack could be performed physically on the object or device.

All 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.)

As a conclusion, if an attacker has physical access to a sensor or actuator device, then much of the security functionality elaborated in this draft is not effective to protect the asset during the physical attack.

Since it does not make sense to design a solution for a situation that cannot be protected against we assume there is no need to protect assets which are exposed during a physical attack. In other words, either an attacker does not have physical access to the sensor or actuator device, or if it has, the attack shall only have effect during the period of physical attack.

5.2 Time Measurements

Measuring time with certain accuracy is important to achieve certain security properties, for example to determine whether a public key certificate, access token or some other assertion is valid.

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 can connect directly to AS it could get updated in terms of time as well as revocation information.

If RS continuously measures time but can’t connect to AS or other trusted source, time drift may have to be accepted and it may not be able to manage revocation. However, it 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 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.
6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgements

The authors would like to thank Carsten Bormann, Stefanie Gerdes, Sandeep Kumar, John Mattson, Corinna Schmitt, Mohit Sethi, Hannes Tschofenig, Vlasios Tsiatsis and Erik Wahlstroem for contributing to the discussion, giving helpful input and commenting on the 00-version. The authors would also like to acknowledge input provided by draft-gerdes-ace-actors [I-D.gerdes-ace-actors] and by Hummen et al. [HUM14delegation].

8. References

8.1 Informative References


Authors' Addresses
Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se

Goeran Selander
Ericsson
Farogatan 6

Seitz & Selander       Expires September 10, 2015
Abstract

This memo presents a scheme for data object security applicable to protection of payload of generic message formats as well as request and response messages of the Constrained Application Protocol (CoAP).

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License 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.
Table of Contents

1. Introduction ................................................. 4
  1.1 Terminology ............................................. 4
2. Background .................................................. 5
4. Secure Message .............................................. 7
  4.1 Secure Message format ..................................... 8
    4.1.1 Secure Message Header .................................. 8
    4.1.2 Secure Message Body ................................... 9
      4.1.2.1 Secure Signed Message Body ......................... 9
      4.1.2.2 Secure Encrypted Message Body .................... 9
    4.1.3 Secure Message Tag .................................... 9
5. Message Protection .......................................... 9
  5.1 CoAP Message Protection .................................. 10
    5.1.1 The Sig Option ....................................... 10
      5.1.1.1 Option Structure .................................. 10
      5.1.1.2 Integrity Protection and Verification .............. 11
      5.1.1.3 SSM Body ........................................ 11
    5.1.2 The Enc Option ....................................... 12
      5.1.2.1 Option Structure .................................. 12
      5.1.2.2 Encryption and Decryption .......................... 12
      5.1.2.3 SEM Body ......................................... 13
      5.1.2.4 CoAP Message with Enc Option ..................... 13
    5.1.3 SM Tag .............................................. 14
  5.2 Application Layer Protection ................................ 14
  5.3 Replay Protection and Freshness ................................ 15
    5.3.1 Replay Protection .................................... 15
    5.3.2 Freshness .......................................... 15
6. Security Considerations ...................................... 16
7. Privacy Considerations ...................................... 17
8. IANA Considerations ........................................ 18
9. Acknowledgements ........................................... 18
10. References ................................................ 19
  10.1 Normative References ..................................... 19
  10.2 Informative References ................................... 19
Appendix A. Which CoAP Header Fields and Options to Protect ........ 20
  A.1 CoAP Header Fields ...................................... 20
  A.2 CoAP Options ........................................... 20
    A.2.1 Integrity Protection ................................. 21
1. Introduction

The Constrained Application Protocol CoAP [RFC7252] was designed with a constrained RESTful environment in mind. CoAP references DTLS [RFC6347] for securing the message exchanges. Two commonly used features of CoAP are store-and-forward and publish-subscribe exchanges, which are problematic to secure with DTLS and transport layer security. As DTLS offers hop-by-hop security, in case of store-and-forward exchanges it necessitates a trusted intermediary. On the other hand, securing publish-subscribe CoAP exchanges with DTLS requires the use of the keep-alive mechanism which incurs additional overhead and actually takes away most of the benefits of asynchronous communication.

The pervasive monitoring debate has illustrated the need to protect data also from trustworthy intermediary nodes as they can be compromised. The community has reacted strongly to the revelations, and new solutions must consider this attack [RFC7258] and include encryption by default.

This memo presents an object security approach for secure messaging in constrained environments that may be used as a complement to DTLS for store-and-forward and publish-subscribe CoAP exchanges. Note that the solution sketched in this memo can be combined with DTLS thus enabling, for example, end-to-end security of CoAP payload in combination with hop-by-hop protection of the entire CoAP messages during transport between end-point and intermediary node.

This version of the draft focuses on symmetric key based algorithms. Public key based algorithms will be addressed in the next version.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

Certain security-related terms are to be understood in the sense defined in RFC 4949 [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", "signature", and "verify".

RESTful terms, such as "resource" or "representation", are to be understood as used in HTTP [RFC7231] and CoAP.
Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Client, Resource Server, and Authorization Server are defined in [I-D.seitz-ace-problem-description]. The terms "server" and "Resource Server" are used interchangeably.

JSON Web Signature (JWS), JOSE Header, JWS Payload, and JWS Signature are defined in [I-D.ietf-jose-json-web-signature].

JSON Web Encryption (JWE), JWE AAD, JWE Ciphertext, and JWE Authentication Tag are defined in [I-D.ietf-jose-json-web-encryption].

Secure Message (SM), Secure Signed Message (SSM), and Secure Encrypted Message (SEM) are message formats defined in this memo. The Compact Secure Message (CSM) format is defined in Appendix C. The Sig and Enc options are CoAP options defined in this memo.

Excluded Authenticated Data (EAD) is defined in this memo (see Sections 4.1.2). Transaction Identifier (TID) is defined in this memo (see Section 4.1.1).

2. Background

The background for this work is provided by the use cases and problem description in [I-D.ietf-ace-usecases] and [I-D.seitz-ace-problem-description]. The overall objective is that (a) only authorized requests are granted, and (b) messages between client and server are protected (according to requirements of the particular use case). The focus of this memo is on end-to-end security in constrained environments in the presence of intermediary nodes, which corresponds to point (b).

For constrained-node networks there may be several reasons for messages to be cached or stored in one node and later forwarded. For example, connectivity between the nodes may be intermittent, or some node may be sleeping at the time when the message should have been forwarded (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.1). Also, the architectural model or protocol applied may require an intermediary node which breaks security on transport layer (see e.g. [I-D.ietf-ace-usecases] sections 2.1.1, and 2.5.2). Examples of intermediary nodes include forward proxies, reverse proxies, pub-sub brokers, HTTP-CoAP cross-proxies, and SMS servers.

On a high level, end-to-end security in this setting encompasses:

1. Protection against eavesdropping and manipulation of resource
representations in intermediary nodes;

2. Protection against message replay;

3. Protection of authorization information ("access tokens") in transport from an Authorization Server to a Resource Server via a Client, or other intermediary nodes which could gain from changing the information;

4. Allowing a client to verify that a response comes from a certain server and is the response to a particular request;

5. Protection of the RESTful method used by the client, or the response code used by the server. For example if a malicious proxy replaces the client requested GET with a DELETE this must be detected by the server;

6. Protection against eavesdropping of meta-data of the request or response, including CoAP options such as for example Uri-Path and Uri-Query, which may reveal some information on what is requested.

From the listed examples, there are two main categories of security requirements and corresponding solutions. The first category deals essentially with application layer protection, i.e. protecting the payload of the RESTful protocol (1-3). The second category deals with protecting an entire CoAP message, targeting also CoAP options and header fields (4-6). The next section formulates security requirements for the two categories, which are denoted Mode:APPL and Mode:COAP, respectively.

3. End-to-end Security in Presence of Intermediary Nodes

For high-level security requirements related to resource access, see section 4.6 of [I-D.seitz-ace-problem-description]. This section defines the specific requirements that address the two categories of examples identified in the previous section, taking into account potential intermediary nodes.

In the case of application layer protection (Mode:APPL), the end-to-end security requirements apply to the RESTful protocol payload data, such as Resource Representations:

a. The payload shall be integrity protected and should be encrypted end-to-end from sender to receiver.

b. It shall be possible for an intended receiver to detect if it has received this message previously, i.e. replay protection.
In this case there may be multiple receivers of a given message, for example in the case of a proxy that is caching responses used to serve multiple clients, or in a publish-subscribe setting with multiple subscribers to a given publication.

In the case of protecting specific Client-Server CoAP message exchanges (Mode:COAP), potentially passing via intermediary nodes, there are additional end-to-end security requirements:

c. The CoAP options which are not intended to be changed by an intermediary node shall be integrity protected between Client and Server.

d. The CoAP options which are not intended to be read by an intermediary node shall be encrypted between Client and Server.

e. The CoAP header field "Code" shall be integrity protected between Client and Server.

f. A Client shall be able to verify that a message is the response to a particular request the Client made.

The requirements listed above can be met by encryption, integrity protection and replay protection. What differs is the actual data that is protected, i.e. application layer data or CoAP message data. This memo specifies a common "Secure Message" format that can be used to wrap either payload only or also additional selected CoAP message fields, and be sent as part of the message.

4. Secure Message

There exist already standardized and draft content formats for cryptographically protected data such as CMS [RFC5652], JWS, JWE, and COSE [I-D.bormann-jose-cose]. None of the listed formats provide support for replay protection, but it is noted in section 10.10 of [I-D.ietf-jose-json-web-signature] that one way to thwart replay attacks is to include a unique transaction identifier and have the recipient verify that the message has not been previously received or acted upon.

The term Secure Message (SM) format refers to a content format for cryptographically protected data which includes a unique transaction identifier and allows customization to support different variants of format and message processing (Modes).

This memo uses JOSE content formats as a model to specify format and processing of messages. The terms Secure Signed Message (SSM) format...
and Secure Encrypted Message (SEM) format to refer to Secure Message formats supporting integrity protection only and additional encryption, analogous to JWS and JWE, respectively. Appendix B shows how JWS and JWE could be extended to become Secure Message formats.

It should be noted that the current JOSE objects are undesirably large for very constrained devices. In their current size they can lead to packet fragmentation in constrained-node networks due to limited frame sizes, and to problems with limited storage capacity on constrained devices due to limited RAM. COSE renders more compact objects, and further optimizations are considered. See Appendix C for a discussion of minimum message expansion and message format overhead.

4.1 Secure Message format

A Secure Message (SM) SHALL consist of Header, Body and Tag.

4.1.1 Secure Message Header

The following parameters SHALL be included in the SM Header:

- Algorithm. This parameter allows the receiver to identify the cryptographic algorithm(s) used to protect the Secure Message. In case of SSM it has the same syntax as the JOSE Header Parameter "alg" defined in Section 4.1.1 of [I-D.ietf-jose-json-web-signature]. In case of SEM, it has the same syntax as the JOSE Header Parameter "enc" defined in Section 4.1.2 of [I-D.ietf-jose-json-web-encryption]. (Assuming direct key agreement, corresponding to the JWE "alg" = "dir" setting.)

- Key Identifier. This parameter allows the receiver to uniquely identify the sender and the security context/key(s) used with the Algorithm. It has the same syntax as the JOSE Header Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature].

- Sequence Number. The Sequence Number parameter enumerates the Secure Messages protected using the key(s) identified by the Key Identifier, and is used for replay protection and uniqueness of nonce. The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

- Mode. The Mode parameter defines application specific message format, content and processing. This parameter provides means for customization of the Secure Message format, in particular to distinguish between Secure Messages containing application layer data only or CoAP message data.
The ordered sequence (Sequence Number, Key Identifier) is called Transaction Identifier (TID), and SHALL be unique for each SM.

4.1.2 Secure Message Body

Analogously to JWS and JWE, the SM Body contains what is being protected. The SM Body is different for SSM and SEM.

In order to obtain a compact representation, certain data is integrity protected but excluded from the Secure Message. Such data is referred to as Excluded Authenticated Data (EAD). To further reduce message size, the unencrypted part of the SM Body may be "detached" from the Secure Message, see sections 4.1.2.1 and 4.1.2.2.

The assumption behind excluding integrity protected data from the SM, or detaching integrity protected but not encrypted parts of the SM during transport, is that the data in question is known to the receiver, e.g. because it is exchanged beforehand or because it is transported as part of the CoAP message carrying the Secure Message.

4.1.2.1 Secure Signed Message Body

For SSM, the Body consists of the payload data which is integrity protected, analogously to the JWS Payload. Detached Content is defined to mean that the Body is removed from the Secure Message, analogously to Appendix F of [I-D.ietf-jose-json-web-signature]. Hence a SSM with Detached Content consists of Header and Tag.

4.1.2.2 Secure Encrypted Message Body

Analogously to JWE, the terms Plaintext, Ciphertext and Additional Authenticated Data (AAD) are used for the SEM. The Body of a SEM consists of Ciphertext and Additional Authenticated Data (AAD). For SEM Detached Content is defined to mean that the AAD is removed from the Secure Message. Hence a SEM with Detached Content consists of the Header, Ciphertext and Tag.

4.1.3 Secure Message Tag

The SM Tag consists of the Signature / Authentication Tag value as defined by the Algorithm, calculated over the SM Header, SM Body and EAD (if present). The content of EAD depends on the Mode, see 5.1.3 and 5.2

5. Message Protection
This section describes what is protected in a Secure Message and how it depends on the defined Modes ("CoAP Message Protection" and "Application Layer Protection"). Both formats SSM and SEM defined in the previous section are applicable to both Modes. For examples, see Appendix D.

For any Secure Message Mode, the SEM format SHALL be used by default.

The SM Header is defined in 4.1.1, indicates the Mode, but is in all other respects handled similarly in both Modes. This section also describes the differences in SM Body and SM Tag.

5.1 CoAP Message Protection

Referring to examples 4-6 in Section 2 and requirements a-f in Section 3, this section presents how to protect individual CoAP messages including options and header fields, as well as request-response message exchanges, using the Secure Message format. This is called Secure Message Mode:COAP. An endpoint receiving a CoAP request containing a Secure Message with Mode:COAP MUST respond with a CoAP message containing a Secure Message with Mode:COAP.

Since slightly different message formats are used for integrity protection only (SSM), and additional encryption (SEM), these cases are treated separately. Two new CoAP security options are introduced: the Enc option and the Sig option. A CoAP message SHALL NOT include both Enc and Sig options.

5.1.1 The Sig Option

In order to integrity protect CoAP message exchanges, a new CoAP option is introduced: the Sig option, containing a SSM Mode:COAP object. Endpoints supporting this scheme MUST check for the presence of a Sig option, and verify the SSM as described in Section 5.1.1.2 before accepting a message as valid.

5.1.1.1 Option Structure

The Sig option indicates that certain CoAP Header Fields, Options, and Payload (if present) are integrity and replay protected using a Secure Signed Message (SSM). The Sig option SHALL contain a SSM with Detached Content (see Section 4.1.2.1).

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 1 illustrates the structure of this option.
<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>Sig</td>
<td>opaque</td>
<td>12-TBD</td>
</tr>
</tbody>
</table>

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 1: The Sig Option

*) Length is essentially Length(SSM Header) + Length(SSM Tag). The
minimum length is estimated in Appendix C. The maximum length
depends on actual message format selected and is TBD.

5.1.1.2 Integrity Protection and Verification

A CoAP endpoint composing a message with the Sig option SHALL process
the SSM and produce the SSM Tag, as defined in 5.1.1.3 and 5.1.3,
alogously to the specification for producing a JWS object as
described in Section 5.1 of [I-D.ietf-jose-json-web-signature] (cf.
Appendix B). In addition, the sending endpoint SHALL process the
Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Sig option SHALL
first recreate the SSM Body as described in Section 5.1.1.3, and then
verify the SSM Tag as described in Section 5.1.3, analogously to the
specification for verifying a JWS object as described in Section 5.2
of [I-D.ietf-jose-json-web-signature] (cf. Appendix B). In addition,
the receiving endpoint SHALL process the Sequence Number as described
in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure
will be included in a future version of this draft.

5.1.1.3 SSM Body

The SSM Body SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;

- all CoAP options present which are marked as IP in Table 3
  (Appendix A), in the order as given by the option number (each
  Option with Option Header including delta to previous IP-marked
  Option which is present); and

- the CoAP Payload (if any).
5.1.2 The Enc Option

In order to encrypt and integrity protect CoAP messages, a new CoAP option is introduced: the Enc option, indicating the presence of a SEM Mode:COAP object in the CoAP message, containing the encrypted part of the CoAP message. Endpoints supporting this scheme MUST check for the presence of an Enc option, and verify the SEM as described in 5.1.2.2 before accepting a message as valid.

NOTE: This version of the draft is only considering AEAD algorithms.

5.1.2.1 Option Structure

The Enc option indicates that certain CoAP Options and Payload (if present) are encrypted, integrity and replay protected using a Secure Encrypted Message (SEM) with Detached Content (see Section 4.1.2.2). The structure of a CoAP message with an Enc option is described in Section 5.1.2.4.

This option is critical, safe to forward, it is not part of a cache key, and it is not repeatable. Table 2 illustrates the structure of this option.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>Enc</td>
<td>opaque</td>
<td>0 or 12-TBD</td>
</tr>
</tbody>
</table>

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 2: The Enc Option

*) Length indicates in this case the additional length added to the total length of all CoAP options. If the CoAP message has Payload, then the Enc option is empty, otherwise it contains the SEM (see Section 5.1.2.4). In the latter case, the SEM Ciphertext contains the encrypted CoAP Options (see Section 5.1.2.3), which are thus excluded from plaintext part of the message. Hence the additional length is essentially Length(SEM Header) + Length(SEM Tag). The minimum length is estimated in Appendix C. The maximum length depends on actual message format selected and is TBD.

5.1.2.2 Encryption and Decryption

A CoAP endpoint composing a message with the Enc option SHALL process
the SEM and produce the SEM Ciphertext and SEM Tag, as defined in 5.1.2.3 and 5.1.3, analogously to the specification for producing a JWE object as described in Section 5.1 of [I-D.ietf-jose-json-web-encryption] (cf. Appendix B). In addition, the sending endpoint SHALL process the Sequence Number as described in Section 5.3.

A CoAP endpoint receiving a message containing the Enc option SHALL first recreate the SEM Body as described in Section 5.1.2.3, and then decrypt and verify the SEM analogously to the specification for verifying a JWE object as described in Section 5.2 of [I-D.ietf-jose-json-web-encryption] (cf. Appendix B). In addition, the receiving endpoint SHALL process the Sequence Number as described in Section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

5.1.2.3 SEM Body

The SEM Plaintext SHALL consist of the following data, formatted as a CoAP message without Header consisting of:

- all CoAP Options present which are marked as E in Table 3 (see Appendix A), in the order as given by the Option number (each Option with Option Header including delta to previous E-marked Option); and

- the CoAP Payload, if present, and in that case prefixed by the one-byte Payload Marker (0xFF).

The SEM Additional Authenticated Data SHALL consist of the following data, in this order:

- the 8-bit CoAP header field Code;

- all CoAP options present which are marked as IP and not marked as E in Table 2 (see Appendix A), in the order as given by the Option number (each Option with Option Header including delta to previous such Option).

5.1.2.4 CoAP Message with Enc Option

An unprotected CoAP message is encrypted and integrity protected by means of an Enc option and a SEM. The structure and format of the protected CoAP message being sent instead of the unprotected CoAP message is now described.
The protected CoAP message is formatted as an ordinary CoAP message, with the following Header, Options and Payload:

- The CoAP header SHALL be the same as the unprotected CoAP message.
- The CoAP options SHALL consist of the unencrypted options of the unprotected CoAP message, and the Enc option. The options SHALL be formatted as in a CoAP message (each Option with Options Header including delta to previous unencrypted Option).
- If the unprotected CoAP message has no Payload then the Enc option SHALL contain the SEM with Detached Content. If the unprotected CoAP message has Payload, then the SEM option SHALL be empty and the Payload of the CoAP message SHALL be the SEM with Detached Content. The Payload is prefixed by the one-byte Payload Marker (0xFF).

5.1.3 SM Tag

This section describes the SM Tag for Mode:COAP, which applies both to SEM and SSM. The SM Tag is defined in 4.1.3. If the message is a CoAP Request, then EAD SHALL be empty. If the message is a CoAP Response, then EAD SHALL consist of the TID of the associated CoAP Request.

5.2 Application Layer Protection

Referring to examples 1-3 in Section 2 and requirements a and b in Section 3, the case of only protecting Payload sent in a RESTful protocol using the Secure Message format is now discussed. This is called Secure Message Mode:APPL.

The sending endpoint SHALL wrap the Payload, and the receiving endpoint unwrap the Payload in the relevant SM format (SSM or SEM) Mode:APPL. The SSM (SEM) SHALL be protected (encrypted) and verified (decrypted) as described in 5.1.1.2 (5.1.2.2), including replay protection as described in section 5.3.

NOTE: The explicit steps of the protection and verification procedure will be included in a future version of this draft.

For Mode:APPL, the EAD SHALL be empty. Hence, the SM Tag is calculated over the SM Header and SM Body.

A CoAP message where the Payload is wrapped as a Secure Message
Mode:APPL object is indicated by setting the option Content-Format to application/sm. A CoAP client may request a response containing such a payload wrapping by setting the option Accept to application/sm. (See Section 8.)

5.3 Replay Protection and Freshness

In order to protect from replay of messages and verify freshness of responses, a CoAP endpoint SHALL maintain Transaction Identifiers (TIDs) of sent and received Secure Messages (see section 4.1.1).

5.3.1 Replay Protection

An endpoint supporting Secure Message SHALL maintain two TIDs and associated security context/key(s) for each other endpoint it communicates with, one TID for protecting sent messages, and one TID for verifying the received messages. Depending on use case, an endpoint MAY maintain a sliding receive window for Sequence Numbers associated to TIDs in received messages, equivalent to the functionality described in section 4.1.2.6 of [RFC6347].

Before composing a new message a sending endpoint supporting Secure Message SHALL step the Sequence Number of the associated send TID and SHALL include it in the SM Header parameter Sequence Number as defined in section 4.1.1. However, if the Sequence Number counter wraps, the client must first acquire a new TID and associated security context/key(s). The latter is out of scope of this memo.

A receiving endpoint supporting Secure Message SHALL verify that the Sequence Number received in the SM Header is greater than the Sequence Number in the TID for received messages (or within the sliding window and not previously received) and update the TID (window) accordingly.

5.3.2 Freshness

If a CoAP server receives a valid Secure Message request in Mode:COAP, then the response SHALL include the TID of the request as EAD, as defined in section 5.1.3. If the CoAP client receives a Secure Message response in Mode:COAP, then the client SHALL verify the signature by reconstructing SM Body and using the TID of its own associated request as EAD, as defined in section 5.1.3.
6. Security Considerations

In scenarios with proxies, gateways, or caching, DTLS only protects data hop-by-hop meaning that all intermediary nodes can read and modify information. The trust model where all participating nodes are considered trustworthy is problematic not only from a privacy perspective but also from a security perspective as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture weak.

DTLS protects the entire CoAP message including Header, Options and Payload, whereas this proposal only protects selected message fields. DTLS, however, also incurs a large overhead cost, e.g. due to the handshake procedure. While that cost can be amortized in scenarios with long lived connections, in cases where a device will have connections with varying clients, using secured objects instead of session security can provide a significant performance gain.

Secure Message Mode: COAP addresses point to point encryption, integrity and replay protection, and freshness of response. Payload as well as relevant options and header field Code are protected. It is possible to define unique session keys to enable perfect forward secrecy.

Secure Message Mode: APPL only protects payload and only gives replay protection (not freshness), but this allows more use cases such as point to multi-point including publish-subscribe, reverse proxies and proxy caching of responses. In case of symmetric keys the receiver does not get data origin authentication, which requires a digital signature using a private asymmetric key.

Using blockwise transfer [I-D.ietf-core-coap-block], the integrity protection as provided by the method described here only covers the individual blocks, not the entire request or response. One way to handle this would to allow the Sig or Enc option to be repeatable, and in one or several of the block transfer carry a MAC or signature that covers the entire request or response.

The Version header field is not integrity protected to allow backwards compatibility with future versions of CoAP. Considering this, it may in theory be possible to launch a
cross-version attack, e.g. something analogously to a bidding down attack. Future updates of CoAP would need to take this into account.

The use of sequence numbers for replay protection introduces the problem related to wrapping of the counter. The alternatives also have issues: very constrained devices may not be able to support accurate time or generate and store large numbers of random nonces. The requirement to change key at counter wrap is a complication, but it also forces the user of this specification to think about implementing key renewal.

Independently of message format, and whether the target is application layer protection or CoAP message protection, this specification needs to be complemented with a procedure whereby the client and the server establish the keys used for wrapping and unwrapping the Secure Message. One way to address key establishment is to assume that there is a trusted third party which can support client and server, such as the Authorization Server in [I-D.seitz-ace-problem-description]. The Authorization Server may, for example, authenticate the client on behalf of the server, or provide cryptographic keys or credentials to the client and/or server which can be used in the Secure Message exchange.

The security contexts required for SSM and SEM are different. For a SSM, the security context is essentially Algorithm, Key Identifier, Sequence Number and Key. For a SEM it is also required to have a unique AEAD Initialization Vector for each message. The AEAD Initialization Vector SHALL be the concatenation of a Salt (8 bytes unsigned integer) and the Sequence Number. The Salt SHOULD be established between sender and receiver before the message is sent, to avoid the overhead of sending it. For example, the Salt may be established by the same means as the keys used to secure the protocol between the sender and receiver. For a SEM, the security context is essentially Algorithm, Key Identifier, Salt, Sequence Number and Key.

NOTE: This last paragraph will be moved into the main document in a future version of this draft.

7. Privacy Considerations

End-to-end integrity protection provides certain privacy properties, e.g. protection of communication with sensor and actuator from manipulation which may affect the personal sphere. End-to-end encryption of payload and certain options provides...
additional protection as to the content and nature of the message exchange.

The headers sent in plaintext allows for example matching of CON and ACK (CoAP Message Identifier), matching of request and response (Token). Plaintext options could also reveal information, e.g. lifetime of measurement (Max-age), or that this message contains one data point in a sequence (Observe).

8. IANA Considerations

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

The following entry is added to the CoAP Option Numbers registry:

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>Sig</td>
<td>[this document]</td>
</tr>
<tr>
<td>TBD</td>
<td>Enc</td>
<td>[this document]</td>
</tr>
</tbody>
</table>

NOTE: IANA considerations for Mode is TBD

This document registers the following value in the CoAP Content Format registry established by [RFC7252].

Media Type: application/sm

Encoding: -

Id: 70

Reference: [this document]

9. Acknowledgements

Klaus Hartke has independently been working on the same problem and a similar solution: establishing end-to-end security across proxies by adding a CoAP option. The authors would like to
thank Francesca Palombini for contributing to the discussion and giving helpful implementation input to the specification. We are grateful to Malisa Vucinic for providing many helpful comments.

10. References

10.1 Normative References

[I-D.ietf-jose-json-web-signature]

[I-D.ietf-jose-json-web-encryption]


10.2 Informative References

[I-D.seitz-ace-problem-description]

[I-D.ietf-ace-usecases]

[I-D.bormann-jose-cose]
Bormann, C., "Constrained Object Signing and Encryption
Appendix A. Which CoAP Header Fields and Options to Protect

In the case of CoAP Message Protection (Mode:COAP) as much as possible of the CoAP message is protected. However, not all CoAP header fields or options can be encrypted and integrity protected, because some are intended to be read or changed by an intermediary node.

A.1 CoAP Header Fields

The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length may be changed by a proxy and thus SHALL neither be integrity protected nor encrypted. Example 5 in Section 2 shows that the Code SHALL be integrity protected. The Version parameter SHALL neither be integrity protected nor encrypted (see Section 6).

A.2 CoAP Options
This section describes what options need to be integrity protected and encrypted. On a high level, all CoAP options must be encrypted by default, unless intended to be read by an intermediate node; and integrity protected, unless intended to be changed by an intermediate node.

However, some special considerations are necessary because CoAP defines certain legitimate proxy operations, because the security information itself may be transported as an option, and because different processing is performed for SSM and SEM.

A.2.1 Integrity Protection

As a general rule, CoAP options which are Safe-to-Forward SHALL be integrity protected, with the only exception being Enc and Sig, which are the security-providing options.

The Unsafe options are divided in two categories, those that are intended to change in a way that can be reconstructed by the server, and those which are not. The following options are of the latter kind and SHALL NOT be integrity protected: Max-Age, Observe, Proxy-Scheme. These options are intended to be changed by a proxy.

For options related to URI of resource (Uri-Host, Uri-Port, Uri-Path, Uri-Query, Proxy-Uri) a Forward Proxy is intended to replace the Uri-* options with the content of the Proxy-Uri option. These options are Unsafe, but the Forward Proxy is intended to perform this precise operation and we can use this predictability to integrity protect the destination endpoint URI, even if the options where the information elements of the URI is located is changed by the Proxy.

This memo makes the full URI located in option 35 (Proxy-Uri) into a common denominator for the URI integrity, as described in the following. The following processing applies to a SSM, for SEM see next section:

- If there is a Proxy-Uri present, then the client MUST integrity protect the Proxy-Uri option and the Uri-* options MUST NOT be integrity protected.
- If there is no Proxy-Uri option present, then the client SHALL compose the full URI from Uri-* options according to the method described in section 6.5 of [RFC7252]. The SM Tag is calculated on the following message, modified compared to what is sent:
  - All Uri-* options removed
  - A Proxy-Uri option with the full URI included
The server SHALL compose the URI from the Uri-* options according to the method described in section 6.5 of [RFC7252]. The so obtained URI is placed into a Proxy-Uri option (no. 35), which is included in the integrity verification.

A.2.2 Encryption

All CoAP options MUST be encrypted, except the options below which MUST NOT be encrypted:

- Max-Age, Observe: This information is intended to be read by a proxy.
- Enc, Sig: These are the security-providing options.
- Uri-Host, Uri-Port: This information can be inferred from destination IP address and port.
- Proxy-Uri, Proxy-Scheme: This information is intended to be read by a proxy.

In the case of a SEM, the Proxy-Uri MUST only contain Uri-Host and Uri-Port and MUST NOT contain Uri-Path and Uri-Query because the latter options are not intended to be revealed to a Forward Proxy.

A.2.3 Summary

Table 3 summarizes which options are encrypted and integrity protected, if present.

In a SSM, options marked with "a" and "b" are composed into a URI as described above and included as the Proxy-Uri option which is part of the SSM Body. In a SEM, options marked "a" are composed into a URI as described above and included as the Proxy-Uri option in the SEM Additional Authenticated Data.

<table>
<thead>
<tr>
<th>No.</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Format</th>
<th>Length</th>
<th>E</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>If-Match</td>
<td>opaque</td>
<td>0-8</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
<td>Uri-Host</td>
<td>string</td>
<td>1-255</td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>ETag</td>
<td>opaque</td>
<td>1-8</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>If-None-Match</td>
<td>empty</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>x</td>
<td>-</td>
<td></td>
<td>Observe</td>
<td>uint</td>
<td>0-3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix B. JOSE Objects as Secure Messages

This section shows how to extend JWS and JWE to Secure Message formats (see Section 4.1). The use of compact serialization is assumed.

B.1 JWS as Secure Signed Message

The JOSE Header of JWS contains the mandatory parameter "alg", defined in Section 4.1.1 of [I-D.ietf-jose-json-web-signature], which corresponds to the parameter Algorithm of the Secure Message.

A JWS is a Secure Message if the JOSE Header includes

- the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature];
- the new Parameter "seq" defined in B.3; and
- the new Parameter "mod" defined in B.4.

In case of JWS, a SSM with Detached Content consists of the JOSE Header and JWS Signature; i.e. no JWS Payload.

B.2 JWE as Secure Encrypted Message

In case of JWE, the SM Header parameters of a JWE consists of the JOSE Header Parameters and JWE Initialization Vector (IV).
The JOSE Header of JWE contains the mandatory parameter "enc", defined in Section 4.1.2 of [I-D.ietf-jose-json-web-encryption], which corresponds to the parameter Algorithm of the Secure Message. The JOSE Header also contains the mandatory parameter "alg", the key encryption algorithm, which in the current version of the draft is assumed to be equal to "dir" (constant). It is also assumed that plaintext compression (zip) is not used.

A JWE is a Secure Message if the IV contains the SM Sequence Number, and the JOSE Header includes

- the Parameter "kid" defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature]; and
- the new Parameter "mod" defined in B.4.

The IV also contain a Salt (see Section 6). For JWE it is mandatory to include the IV and hence the Salt is sent in each message.

In case of JWE, a SEM with Detached Content consists of JOSE Header, JWE Initialization Vector, JWE Ciphertext and JWE Authentication Tag; i.e. no JWE AAD.

B.3 "seq" (Sequence Number) Header Parameter

The Sequence Number SHALL be a 64-bit unsigned integer in hexadecimal representation. Only the significant bytes are sent (initial bytes with zeros are removed). The start sequence number SHALL be 0. For a given key, any Sequence Number MUST NOT be used more than once.

The parameter "seq" SHALL be marked as critical using the "crit" header parameter (see section 4.1.11 of [I-D.ietf-jose-json-web-signature]), meaning that if a receiver does not understand this parameter it must reject the message.

B.4 "mod" (Mode) Header Parameter

The Mode parameter SHALL be an 8-byte unsigned integer defining application specific message format, content and processing. The parameter "mod" SHALL be marked as critical. "mod":"0" indicates Mode:APPL which is defined in Section 5.2. "mod":"1" indicates Mode:COAP which is defined in Section 5.1.

B.4 The TID consists of the concatenation of SEQ and KID, in that order, formatted as in the JOSE. For "seq" the initial bytes with zeros are removed.
Appendix C. Compact Secure Message

For constrained environments it is important that the message expansion due to security overhead is kept at a minimum. As an attempt to assess what this minimum expansion could be, an optimized Secure Message format is defined, tailor-made for this setting. This is intended as a benchmark for generic content formats, to allow an informed decision about which Secure Message format to mandate in a future version of this draft.

C.1 CSM Format

This section defines a compact Secure Message format (see Section 4.1) called the Compact Secure Message (CSM) format, see Figure 4.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| M |    ALG    |   KL    |  SL |             KID               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   SEQ                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Body                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Tag                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Figure 4: Compact Secure Message format
```

The CSM Header (see Section 4.1.1.) consists of 2 bytes of fixed length parameters and two variable length parameters, Key Identifier (KID) and Sequence Number (SEQ). The Header parameters are (compare Table 5):

- **Mode (M).** M=0 indicates Mode:APPL as defined in Section 5.2. M=1 indicates Mode:COAP as defined in Section 5.1. M=2 and M=3 are reserved for future use.

- **Algorithm (ALG).** This parameter consists of an encoding of the ciphersuite used in the Secure Message. The encoding is TBD.

- **KID Length (KL).** This parameter consists of a length indication of the header parameter Key Identifier. The actual length of KID is KL + 1 bytes.

- **SEQ Length (SL).** This parameter consists of a length indication of the header parameter Sequence Number. The actual length of
SEQ is SL + 1 bytes.

- Key Identifier (KID). This parameter identifies the key(s) used to protect the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).

- Sequence Number (SEQ). This parameter consists of the sequence number used by the sender of the Secure Message. Only the significant bytes are sent (initial bytes with zeros are removed).

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameter</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Mode</td>
<td>2 bits</td>
</tr>
<tr>
<td>ALG</td>
<td>Algorithm</td>
<td>6 bits</td>
</tr>
<tr>
<td>KL</td>
<td>Key Identifier Length</td>
<td>5 bits</td>
</tr>
<tr>
<td>SL</td>
<td>Sequence Number Length</td>
<td>3 bits</td>
</tr>
<tr>
<td>KID</td>
<td>Key Identifier</td>
<td>KL + 1: 1-32 bytes</td>
</tr>
<tr>
<td>SEQ</td>
<td>Sequence Number</td>
<td>SL + 1: 1-8 bytes</td>
</tr>
</tbody>
</table>

Table 5: CSM Header Parameters.
The minimum CSM Header is 4 bytes.

The TID consists of the concatenation of SEQ and KID, in that order, formatted as in the CSM format (initial bytes with zeros are removed).

The content of CSM Body depends on whether it is a SSM or a SEM (see Section 4.1.2) which is determined by the Algorithm. This version of the draft focuses on Secure Message with Detached Content. Hence, the SSM Body is empty and the SEM Body consists of the Ciphertext. In the former case, the length of the CSM Body is 0. In the latter case, the length of the CSM Body equals the sum of the lengths of the present CoAP options marked encrypted in Table 3 and the length of the payload of the unprotected CoAP message.

The CSM Tag contains the MAC/Signature as determined from the Algorithm. The length is determined by ALG.
C.2 Comparison of Secure Message sizes

This section gives some examples of overhead incurred with JOSE, the current proposal for COSE at the time of writing (00-draft), and CSM. The goal is not to give exact measurements, but to help the reader appreciate the rough order of magnitude of the overhead involved. COSE seems to be the most promising approach and CSM should be viewed as an attempt to define a lower bound for COSE.

The comparison is complicated further by the fact that algorithms suitable for constrained environments are not supported by JOSE, and thereby not by COSE. This comparison does not consider the ciphertext or signed payload expansion due to Base64url encoding in JWS/JWE. This would increase the overhead of JWS and JWE even more.

The size of the header is shown separately from the size of the authentication tag, since JWS/JWE has no provisions for truncating it, a feature that could easily be added to the JOSE specifications. For CSM the encoding of certain additional algorithms is assumed and this could also easily be added to COSE. An 8-byte kid is used throughout all examples. Finally compact serialization for both JWS and JWE is assumed.

SSM uses HMAC-SHA256, with truncation to 16 bytes.

For JWS the following header is used:

```json
{"alg":"HS256", "kid":"a1534e3c5fd09bd", "seq":"00000142", "mod":"0"}
```

which encodes to a size of 90 bytes in Base64url, and the 32 bytes of HS256 MAC encode to 43 bytes. The concatenation marks add 2 bytes to that in the total overhead.

The same header in COSE, representing the "kid" as bytes (not as string) and the "seq" as positive integer encodes to a size of 35 bytes, and the MAC would add to 32 bytes to that. Note that encoding the header and the MAC together incurs an additional overhead of 3 bytes.

For CSM the same header is represented by 12 bytes. The MAC could in this case safely be truncated to 16 bytes, and a corresponding algorithm identifier would need to be defined in the list of supported algorithms.

Table 6 summarizes these results.
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>MAC</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWS</td>
<td>90 B</td>
<td>43 B</td>
<td>135 bytes</td>
</tr>
<tr>
<td>COSE</td>
<td>35 B</td>
<td>32 B</td>
<td>70 bytes</td>
</tr>
<tr>
<td>CSM</td>
<td>12 B</td>
<td>16 B</td>
<td>28 bytes</td>
</tr>
</tbody>
</table>

Table 6: Comparison of JWS, COSE, and CSM

For SEM the use of AES-128-CCM-8 would be ideal, but since this is not supported by JOSE, AES-128-GCM is used there instead.

For JWE it is assumed that the IV is generated from the sequence number and some previously agreed upon Salt. This means it is not required to explicitly send the IV in the CSM format, but also that the JWE and COSE formats can omit the sequence number.

The JWE header

{"alg":"dir", "kid":"a1534e3c5fdc09bd", "enc":"A128GCM", "mod":"0"}

encodes to a size of 86 bytes in Base64url, while the necessary 12 byte IV for GCM mode is expanded to 16 bytes by encoding. The 16 bytes of the authentication tag expand to 22 bytes. The concatenation marks add 3 bytes to the total overhead.

In COSE the same header encodes to 40 bytes and the IV and authentication tag could be represented as 12 and 16 bytes respectively. Note that encoding the header, the IV and the authentication tag together incurs an additional overhead of 2 bytes.

For CSM this tests uses CCM mode instead of GCM. CCM requires a 16 byte IV, but is better suited for constrained devices, and for CSM there is no impact since the IV can be deduced from the sequence number and a previously agreed upon Salt. The corresponding header for AES-128-CCM-8, including the 8 byte sequence number, is represented by 12 bytes.

Table 7 summarizes these results.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Header</th>
<th>IV</th>
<th>Tag</th>
<th>Total Overhead</th>
</tr>
</thead>
</table>

Table 7: Comparison of JWE, COSE, and CSM

<table>
<thead>
<tr>
<th></th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>JWE</td>
<td>127</td>
</tr>
<tr>
<td>COSE</td>
<td>70</td>
</tr>
<tr>
<td>CSM</td>
<td>20</td>
</tr>
</tbody>
</table>

Appendix D. Examples

This section gives examples of how to use the new options and message formats defined in this memo.

D.1 CoAP Message Protection

This section illustrates the Secure Message Mode:COAP. The message exchange assumes there is a security context established between client and server. One key is used for each direction of the message transfer. The intermediate node detects that the CoAP message contains a SM Mode:COAP object (Sig or Enc option is set) and thus forwards the message as it cannot serve a cached response.

D.1.1 Integrity protection of CoAP Message

Here is an example of a PUT request/response message exchange passing an intermediate node protected with the Sig option. The example illustrates a client opening a lock and getting a confirmation that the lock is opened. Code, Uri-Path and Payload are integrity protected (see Appendix A).

Client  Proxy  Server

| +----> | PUT  |
|       |      |
| Code: 0.03 (PUT) |
| Token: 0x8c |
| Uri-Path: lock |
| Sig: SSM {"mod":"1","seq":"00000142", "kid":"a1534e3c5fde09bd", ...} |
| Payload: 1 |

| +----> | PUT  |
|       |      |
| Code: 0.03 (PUT) |
| Token: 0x7b |
| Uri-Path: lock |
| Sig: SSM {"mod":"1","seq":"00000142", ...} |
The Key Identifier is a hint to the receiver indicating which security context was used to integrity protect the message, and may be used as an identifier for a secret key or a public key. (It may e.g. be the hash of a public key.)

The server and client can verify that the Sequence Number has not been received and used with this key before, and since Mode is COAP, the client can additionally verify the freshness of the response, i.e. that the response message is generated as an answer to the received request message (see Section 5.3).

The SSM also contains the Tag as specified in the Algorithm (not shown).

This example deviates from encryption (SEM) by default (see Section 6) just to illustrate the Sig option. If there is no compelling reason why the CoAP message should be in plaintext, then the Enc option must be used.

D.1.2 Encryption of CoAP Message

Here is an example of a GET request/response message exchange passing an intermediate node protected with the Enc option. The example illustrates a client requesting a blood sugar measurement resource (GET /glucose) and receiving the value 220 mg/dl. Uri-Path and Payload are encrypted and integrity protected. Code is integrity protected only (see Appendix A).

| Client | Proxy | Server |

Figure 8: CoAP PUT protected with Sig/SSM (Mode:COAP)
Figure 9: CoAP GET protected with Enc/SEM (Mode:COAP).
The bracket [ ... ] indicates encrypted data.

Since the request message (GET) does not support payload, the SEM is
carried in the Enc option. Since the response message (Content)
supports payload, the Enc option is empty and the SEM is carried in
the payload.

The Key Identifier is a hint to the receiver indicating which
security context was used to encrypt and integrity protect the
message, and may be used as an identifier for the AEAD secret key.
One key is used for each direction of the message transfer.

The server and client can verify that the Sequence Number has not
been received and used with this key before, and since Mode:COAP the
client can additionally verify the freshness of the response, i.e.
that the response message is generated as an answer to the received
request message (see Section 5.3).
The SEM also contains the Tag as specified by the Algorithm (not shown).

D.2 Application Layer Protection

This section gives examples that illustrate Secure Message Mode:APPL. This mode assumes that only the intended receiver(s) has the relevant security context related to the resource.

D.2.1 Proxy Caching

This example outlines how a proxy forwarding request and response of one client can cache a response whose payload is a SEM object, and serve this response to another client request, such that both clients can verify integrity and non-replay.

Client1 Proxy Server

\[
\begin{align*}
\text{Client1} & \quad \text{Proxy} \quad \text{Server} \\
\text{---+} & \quad \text{---+} \\
\text{Categories:} & \quad \text{Categories:} \\
\text{+---->} & \quad \text{+---->} \\
\text{GET} & \quad \text{GET} \\
\text{Token:} & \quad \text{Token:} \\
\text{0x83} & \quad 0xbe \\
\text{Proxy-Uri:} & \quad \text{Uri-Host:} \\
\text{example.com/temp} & \quad \text{example.com} \\
\text{---+} & \quad \text{---+} \\
\text{Categories:} & \quad \text{Categories:} \\
\text{<-----} & \quad \text{<-----} \\
\text{Code:} & \quad \text{Code:} \\
\text{2.05} & \quad 2.05 \\
\text{Token:} & \quad \text{Token:} \\
\text{0xbe} & \quad 0x83 \\
\text{Payload:} & \quad \text{Payload:} \\
\text{SEM {"mod":"0","seq":"000015b7",} & \quad \text{SEM {"mod":"0","seq":"000015b7",} \\
\text{"kid":"c09bda155fd34e3c",} & \quad \text{"kid":"c09bda155fd34e3c",} \\
\text{"471 F", ...}} & \quad \text{"471 F", ...}} \\
\text{---+} & \quad \text{---+} \\
\text{Categories:} & \quad \text{Categories:} \\
\text{<-----} & \quad \text{<-----} \\
\text{Code:} & \quad \text{Code:} \\
\text{2.05} & \quad 2.05 \\
\text{Token:} & \quad \text{Token:} \\
\text{0x83} & \quad 0x83 \\
\text{Payload:} & \quad \text{Payload:} \\
\text{SEM {"mod":"0","seq":"000015b7",} & \quad \text{SEM {"mod":"0","seq":"000015b7",} \\
\text{"kid":"c09bda155fd34e3c",} & \quad \text{"kid":"c09bda155fd34e3c",} \\
\text{"471 F", ...}} & \quad \text{"471 F", ...}}
\end{align*}
\]
D.2.2 Publish-Subscribe

This example outlines a publish-subscribe setting where the payload is integrity and replay protected end-to-end between Publisher and Subscriber. The example illustrates a subscription registration and a new publication of birch pollen count of 300 per cubic meters. The PubSub Broker can define the Observe count arbitrarily (as could any intermediary node, even in Mode:COAP), but cannot manipulate the Sequence Number without being noticed.

Sub-    PubSub- Publisher
    scriber  Broker

| +-----+                     Code: 0.01 (GET)  |
| GET   |  Token: 0x72              |
|       |  Uri-Path: ps              |
|       |  Uri-Path: birch-pollen    |
|       |  Observe: 0 (register)     |

| <------  Code: 2.05 (Content) |
|  2.05    |
|          |  Token: 0x72              |
|          |  Observe: 1               |
|          |  Payload: SSM {"mod"="0","seq":"000015b7", "kid":"c09bda155fd34e3c", ["270"], ...} |

| <------  Code: 0.03 (PUT) |
|  PUT     |
|          |  Token: 0x1f              |
|          |  Uri-Path: ps              |
|          |  Uri-Path: birch-pollen    |
|          |  Payload: SSM {"mod"="0","seq":"000015b8",} |
This example deviates from encryption (SEM) by default (see Section 6)just to illustrate the SSM in Mode:APPL. If there is no compelling reason why the payload should be in plaintext, then SEM must be used.

D.2.3 Transporting Authorization Information

This example outlines the transportation of authorization information from a node producing (Authorization Server, AS) to a node consuming (Resource Server, RS) such information. Authorization information may for example be an authorization decision with respect to a Client (C) accessing a Resource to be enforced by RS. See Section 4.4-4.5 of [I-D.seitz-ace-problem-description].

Here, C is clearly not trusted with modifying the information, but may need to be involved with mediating the authorization information to the RS, for example, because AS and RS does not have direct connectivity. So end-to-end security is required and object security is a natural candidate (cf. "Access Tokens").

This example considers the authorization information to be encapsulated in a SEM Mode:APPL object, generated by AS. How C accesses the SSM is out of scope for this example, it may e.g. be using CoAP. C then requests RS to configure the authorization information in the SEM by doing PUT to /authorization. This particular resource has a default access policy that only new messages signed by AS are authorized. RS thus verifies the integrity and sequence number by using the existing security context for the AS, and responds accordingly, a) or b), see Figure 12.
Figure 12: Protected Transfer of Access Token = SEM (Mode:APPL)

Authors' Addresses

Goeran Selander  
Ericsson  
Farogatan 6  
16480 Kista  
SWEDEN  
EMail: goran.selander@ericsson.com

John Mattsson  
Ericsson  
Farogatan 6  
16480 Kista  
SWEDEN  
EMail: john.mattsson@ericsson.com

Ludwig Seitz  
SICS Swedish ICT AB  
Schelevagen 17  
22370 Lund  
SWEDEN