ACE Working Group
Internet-Draft
Intended status: Informational
Expires: October 31, 2015

S. Gerdes
Universitaet Bremen TZI
L. Seitz
SICS Swedish ICT AB
G. Selander
Ericsson
C. Bormann, Ed.
Universitaet Bremen TZI
April 29, 2015

An architecture for authorization in constrained environments
draft-gerdes-ace-actors-05

Abstract

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

This document provides terminology, and elements of an architecture / a problem statement, for authentication and authorization in these networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Constrained nodes are small devices with limited abilities which in many cases are made to fulfill a specific 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.

In some cases authentication and authorization can be addressed by static configuration provisioned during manufacturing or deployment by means of fixed trust anchors and access control lists. This is particularly applicable to siloed, fixed-purpose deployments. However, as the need for flexible access to assets already deployed increases, the legitimate set of authorized entities as well as their privileges cannot be conclusively defined during deployment, without any need for change during the lifetime of the device. Moreover, several use cases illustrate the need for fine-grained access control policies, for which the access control lists concept may not be sufficiently generic.

The limitations of the constrained nodes ask for security mechanisms which take the special characteristics of constrained environments into account; not all constituents may be able to perform all necessary tasks by themselves. In order to meet the security requirements in constrained scenarios, the necessary tasks need to be assigned to logical functional entities.

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

1.1. Terminology

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

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

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

In addition, this document uses the following terminology:

Resource (R): an item of interest which is represented through an interface. It might contain sensor or actuator values or other information.

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

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

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

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

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

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 the requests a Client makes and its acceptance of responses.

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

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

Authenticated Authorization: A synthesis of mechanisms for authentication and authorization.

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

2. Architecture and High-level Problem Statement

2.1. Elements of an Architecture

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

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

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

More on the security objectives of the constrained level in Section 5.1.

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

Figure 1: Constrained Level

The authorization decisions at the endpoints are made on behalf of the principals that control the endpoints. To reuse OAuth and UMA terminology, the present document calls C’s controlling principal the Requesting Party (RqP), and calls RS’s controlling principal the

Resource Owner (RO). Each principal makes authorization decisions (possibly encapsulating them into security policies) which the endpoint it controls then enforces.

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

- o Objectives of type 1: No entity not authorized by the RO has access to (or otherwise gains knowledge of) R.

- o Objectives of type 2: C is exchanging information with (sending a request to, accepting a response from) a resource only where it can ascertain that RqP has authorized the exchange with R.

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

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

```
  -------                           -------
  | RqP |                           |  RO | Principal Level
  -------                           -------
     |                                 |
in charge of                      in charge of
     |                                 |
     V                                 V
  -------                           -------
  |  C  |  -- requests resource --> | RS  | Constrained Level
  -------  <-- provides resource--  -------
```

Figure 2: Constrained Level and Principal Level

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, are not able to communicate directly with the device because of a lack of user interfaces or displays, or may prefer the device to communicate autonomously.

Moreover, constrained endpoints may need support with tasks requiring heavy processing, large memory or storage, or interfacing to humans, such as management of security policies defined by a principal. The principal, in turn, requires some agent maintaining the policies governing how its endpoints will interact.

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

```
 -------                            -------
 | RqP |                            |  RO | Principal Level
 -------                            -------
    |                                  |
controls                           controls
    |                                  |
    V                                  V
--------                           -------
|  CAS |  <- AuthN and AuthZ ->    |  AS |  Less-Constrained Level
--------                           -------
    |                                  |
controls and supports        controls and supports
authentication               authentication
and authorization            and authorization
    |                                  |
    V                                  V
 -------                            -------
 |  C  |  -- requests resource --> | RS  | Constrained Level
 -------  <-- provides resource--   -------
```

Figure 3: Overall architecture

2.2. Architecture Variants

The elements of the architecture described above are architectural. In a specific scenario, several elements 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:

```
      -------                           --------
      | RqP |                           |  RO  |   Principal Level
      -------                           --------
        |                                   |
   in charge of                        in charge of
        |                                   |
        V                                   V
   ------------                      --------
   |  CAS + C | <- AuthN and AuthZ ->  |  AS  |  Less-Constrained Level
   ------------                      --------
               ^                           |
                \__                        |
                   \___                 authentication
                       \___             and authorization
        requests resource/ \___          support
        provides resource      \___        |
                                   \___    |
                                       V   V
                                      -------
                                      | RS  | Constrained Level
                                      -------
```

Figure 4: Combined C and CAS

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

```
      -------                              -------
      | RqP |                              |  RO |   Principal Level
      -------                              -------
        |                                    |
   in charge of                         in charge of
        |                                    |
        V                                    V
   ----------                           -----------
   |  CAS   | <- AuthN and AuthZ ->  | RS + AS |  Less-Constrained Level
   ----------                           -----------
        |                              ^
   authentication                 ___/
   and authorization          ___/
   support                ___/ request resource / provides resource
        |            ___/
        V        ___/
     -------    /
     |  C  | <-
     -------
```

Figure 5: Combined AS and RS

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

```
                    ------------
                    | RqP = RO |   Principal Level
                    ------------
                          |
                      in charge of
                          |
                          V
                  --------------
                  |  CAS + AS  |  Less-Constrained Level
                  --------------
                /                 \
               /                   \
   authentication               authentication
   and authorization            and authorization
   support                      support
            /                        \
           V                          V
     -------                          -------
     |  C  | -- requests resource --> | RS  | Constrained Level
     ------- <-- provides resource -- -------
```

Figure 6: CAS combined with AS

2.3. Information flows

In this subsection, we complement the abstracted architecture described above with a discussion of the information flows in scope, mentioning that each endpoint may assume both a client and a server role and that communication may be via intermediaries.

The less-constrained nodes, CAS and AS, control the interactions between the endpoints by supporting the potentially constrained nodes with control information, for example permissions of clients, conditions on resources, attributes of client and resource servers, keys and credentials. The control information may be rather different for C and RS, reflecting the intrinsic asymmetry with C initiating the request for access to a resource, and RS acting on a received request, and C finally acting on the received response.

The information flows are shown in Figure 7. The arrows with control information only indicate origin and destination of information, actual message flow may pass intermediary nodes (both nodes that are identified in the architecture and other nodes).

```
     -------  ------                    -------  ------
     | CAS |  | AS |                    | CAS |  | AS |
     -------  ------                    -------  ------
         |      |                          |      |
         |      |                          |      |
         |      | control                  |      | control
         |      | information              |      | information
         |      |                          |      |
    --------------------               --------------------
    |    |      |     |               |    |      |     |
    |    v      |     |               |    |      v
    | -------  ---------- request  -----------> -------  |
    | | C1  | <---------- response ------------ | RS2 |  |
    | -------   |     |               |    |   -------  |
    |           v     |               |    v            |
    |         -------  |               | -------         |
    |        | RS1 | <---- request ----- | C2  |         |
    |        ------- ----- response ---> -------         |
    |                 |               |                 |
    |   Endpoint 1    |               |   Endpoint 2    |
    --------------------               --------------------
```

Figure 7: Information flows that need to be protected

o We assume that the necessary keys/credentials for protecting the control information between the potentially constrained nodes and

their associated less-constrained nodes are pre-established, for example as part of the commissioning procedure.

- o The messages between the endpoints also need to be protected, potentially end-to-end through intermediary nodes (Section 3.1). Any necessary keys/credentials for protecting the interaction between the endpoints will need to be established and maintained as part of a solution.

2.4. Problem statement

The problem statement for authorization in constrained environments can be summarized as follows:

- o The interaction between potentially constrained endpoints is controlled by control information provided by less-constrained nodes on behalf of the principals of the endpoints.

- o The interaction between the endpoints needs to be secured, as well as the establishment of the necessary keys for securing the interaction, potentially end-to-end through intermediary nodes.

- o The mechanism for transferring control information needs to be secured, potentially end-to-end through intermediary nodes. Pre-established keying material may need to be employed for establishing the keys used to protect these information flows.

3. Security Objectives

The security objectives that are addressed by an authorization solution include 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. These additional objectives are not related to authorization and thus are not in scope of this draft, but may nevertheless be relevant. Non-repudiation and accountability may require authentication on a device level, if it is necessary to determine which device performed an action. In other cases it may be more important to find out who is responsible for the device’s actions.

The security objectives and their relative importance differ for the various constrained environment applications and use cases [I-D.ietf-ace-usecases].

In many cases, one participating party has different security objectives than another. To achieve a security objective of one party, another party may be required to provide a service. E.g., if RqP requires the integrity of representations of a resource R that RS is hosting, both C and RS need to partake in integrity-protecting the transmitted data. Moreover, RS needs to protect any write access to this resource as well as to relevant other resources (such as configuration information, firmware update resources) to prevent unauthorized users from manipulating R.

3.1. End-to-End Security Objectives

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

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

Note that there may also be cases of intermediary nodes that very much partake in the security objectives to be achieved. What is the endpoint to which communication needs end-to-end protection is defined by the use case.

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

4. Authentication and Authorization

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

To determine if an entity is authorized to access a resource, an authentication mechanism is needed. According to the Internet Security Glossary [RFC4949], authentication is "the process of

verifying a claim that a system entity or system resource has a certain attribute value." Examples for attribute values are the ID of a device, the type of the device or the name of its owner.

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

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

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

(As a special case for the authorization of read access to a resource, RS may simply make an encrypted representation available to anyone [OSCAR]. In this case, controlling read access to that resource can be reduced to controlling read access to the key; partially removing access also requires a timely update of the key for RS and all participants still authorized.)

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

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

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

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

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

5. Actors and their Tasks

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

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

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

may be realized as protocols or be internal to such a piece of software.

5.1. Constrained Level Actors

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

C performs the following tasks:

- o Communicate in a secure way (provide for confidentiality and integrity of messages), including access requests.

- o Validate that an entity is an authorized server for R.

RS performs the following tasks:

- o Communicate in a secure way (provide for confidentiality and integrity of messages), including responses to access requests.

- o Validate the authorization of the requester to access the requested resource as requested.

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.

(See Figure 8.)

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

Figure 8: Constrained Level Actors

5.2. 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:

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

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

RO must fulfill the following task:

- o Configure for RS authorization information for accessing R.

(See Figure 2.)

5.3. Less-Constrained Level Actors

Constrained level actors can only fulfill a limited number of tasks and may not have network connectivity all the time. To relieve them from having to manage keys for numerous endpoints and conducting computationally intensive tasks, another 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 server for R. CAS can do that because for C, CAS is the authority for claims about RS.

CAS performs the following tasks:

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

- o Negotiate means for secure communication to communicate with C.

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

AS performs the following tasks:

- o Validate on the server side that an entity has certain attributes.
- o Obtain authorization information about an entity from RS' principal (RO) and provide it to RS.
- o Negotiate means for secure communication to communicate with RS.

6. Kinds of Protocols

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

6.1. Constrained Level Protocols

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

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

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

Constrained devices have only limited storage space and thus cannot store large numbers of keys. This is especially important because constrained networks are expected to consist of thousands of nodes.

Protocols on the constrained level should keep this limitation in mind.

6.1.1. Cross Level Support Protocols

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.

6.2. Less-Constrained Level Protocols

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

On the less-constrained level, HTTP [RFC7230] and Transport Layer Security (TLS) [RFC5246] can be used alongside or instead of CoAP and DTLS. Moreover, existing security solutions for authentication and authorization such as the OAuth web authorization framework [RFC6749] and Kerberos [RFC4120] can likely be used without modifications and there are no limitations for the use of a Public Key Infrastructure (PKI).

7. Elements of a Solution

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

7.1. Authorization

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

- o AS needs to transfer authorization information to RS and CAS needs to transfer authorization information to C.

- o The transferred authorization information needs to follow a defined format and encoding, which must be efficient for constrained devices, considering size of authorization information and parser complexity.

- o C and RS need to be able to verify the authenticity of the authorization information they receive. Here as well, there is a trade-off between processing complexity and deployment complexity.

- o The RS needs to enforce the authorization decisions of the AS, while C needs to abide with the authorization decisions of the CAS. The authorization information might require additional policy evaluation (e.g. matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the constrained actors needs to be adapted to the capabilities of the devices implementing them.

- o Finally, as is indicated in the previous bullet, for a particular authorization decision there may be different kinds of authorization information needed, and these pieces of information may be transferred to C and RS at different times and in different ways prior to or during the client request.

7.2. Authentication

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

- o RS needs to authenticate AS, and C needs to authenticate CAS, to ensure that the authorization information and related data comes from the correct source.

- o CAS and AS may need to to authenticate each other, both to perform the required business logic and to ensure that CAS gets security information related to the resources from the right source.

- o In some use cases RS needs to authenticate some property of C, in order to map it to the relevant authorization information. 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 CAS and AS need to authenticate their communication partner (C or RS), in order to ensure it serves the correct device.

7.3. Communication Security

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

- o Session-based security at transport layer such as DTLS [RFC6347] offers security, including integrity and confidentiality

protection, for the whole application layer exchange. However, DTLS may not provide end-to-end security over multiple hops. Another problem with DTLS is the cost of the handshake protocol, which may be too expensive for constrained devices especially in terms of memory and power consumption for message transmissions.

- o An alternative is object security at application layer, 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.

7.4. Cryptographic Keys

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

Symmetric vs Asymmetric Keys
: We need keys both for protection of resource access and for protection of transport of authentication and authorization information. 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 7.1 there must be a mapping between these keys and the authorization information, at least in the sense that AS must be able to specify a unique client identifier which RS can verify (using an associated key). One of the use cases of [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?

8. Assumptions and Requirements

In this section we list a set of candidate assumptions and requirements to make the problem description in the previous sections more concise and precise.

8.1. Architecture

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

- o RS hosting resources, and responding to access requests
- o C requesting access to resources
- o AS supporting the access request/response procedure by providing authorization information to RS
  - * AS may support this by aiding RS in authenticating C, or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- o CAS supporting the access request/response procedure by providing authorization information to C
  - * CAS may support this by aiding C in authenticating RS, forwarding information between AS and C (possibly ultimately for RS), or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- o The architecture allows for intermediary nodes between any pair of C, RS, AS, and CAS, such as forward or reverse proxies in the CoRE architecture. (Solutions may or may not support all combinations.)
  - * The architecture does not make a choice between session based security and data object security.

8.2. Constrained Devices

- o C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover:
  - * unable to manage complex authorization policies

  * unable to manage a large number of secure connections

  * without user interface

  * without constant network connectivity

  * unable to precisely measure time

  * required to save on wireless communication due to high power consumption

- o CAS and AS are not assumed to be constrained devices.

- o All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.

  * 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.

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

- o 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.

- o A solution will need to consider support for a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. section Section 9.2).

8.3. Authentication

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

- o Similary, C needs to authenticate CAS to ensure that the authorization information and related data comes from the correct source.

- o Depending on use case and authorization requirements, C, RS, CAS, or AS may need to authenticate messages from each other.

8.4. Server-side Authorization

- o RS enforces authorization for access to a resource based on credentials presented by C, the requested resource, the REST method, and local context in RS at the time of the request, or on any subset of this information.
- o The credentials presented by C may have been provided by CAS.
- o The underlying authorization decision is taken either by AS or RS.
- o The authorization decision is enforced by RS.
  - * RS needs to have authorization information in order to verify that C is allowed to access the resource as requested.
  - * RS needs to make sure that it provides resource access only to authorized clients.
- o Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (e.g. resource descriptions).
- o The solution may need to be able to support the delegation of access rights.

8.5. Client-side Authorization Information

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

8.6. Server-side Authorization Information

- o Authorization information is transferred from AS to RS using Agent, Push or Pull mechanisms [RFC2904].
- o RS needs to authenticate that the authorization information is coming from AS (integrity).
- o The authorization information may also be encrypted end-to-end between AS and RS (confidentiality).
- o The architecture supports the case where RS may not be able to communicate with AS at the time of the request from C.

- o RS may store or cache authorization information.

- o Authorization information may be pre-configured in RS.

- o Authorization information stored or cached in RS needs to be possible to change. The change of such information needs to 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.

8.7. Resource Access

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

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

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

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

- o C needs to be able to verify that the response to a request comes from the intended RS.

- o There may be resources whose access need not be protected (e.g. for discovery of the responsible AS).

8.8. Keys and Cipher Suites

- o A constrained node and its authorization manager (i.e., RS and AS, and C and CAS) have established cryptographic keys. For example, they share a secret key or each have the other’s public key.

- o The transfer of authorization information is protected with symmetric and/or asymmetric keys.

- o The access request/response 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 and to protect the response.

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

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

8.9. Network Considerations

- o A solution will need to consider network overload due to avoidable communication of a constrained node with its authorization manager (C with CAS, RS with AS).

- o A solution will need to consider network overload by compact authorization information representation.

- o A solution may want to optimize the case where authorization information does not change often.

- o A solution may consider support for an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

8.10. Legacy Considerations

- o A solution may consider interworking with existing infrastructure.

- o A solution may consider supporting authorization of access to legacy devices.

9. Security Considerations

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

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. Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications

of specific constraints on the security mechanisms employed. A wider view of security in constrained-node networks is provided in [I-D.garcia-core-security].

9.1. 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:

- o Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.

- o Denial-of-service making the sensor/actuator unable to perform its intended task correctly.

  A number of attacks can be made with physical access to a device including probing attacks, timing attacks, power attacks, etc. However, with physical access to a sensor or actuator device it is possible to directly perform attacks equivalent of eavesdropping, manipulating data or denial of service. For example:

- o 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.

- o 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.

- o 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 full 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, and shall be limited in extent to the physical control the attacker exerts (e.g., must not affect the security of other devices.)

9.2. 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.

10. IANA Considerations

This document has no actions for IANA.

11. 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 specifically acknowledge input provided by Hummen and others [HUM14delegation].

12. Informative References

[HUM14delegation]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", 11th IEEE International Conference on Sensing, Communication, and Networking (SECON’14), June 30 - July 3, 2014.

[I-D.garcia-core-security]
Garcia-Morchon, O., Kumar, S., Keoh, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-06 (work in progress), September 2013.

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-13 (work in progress), April 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-03 (work in progress), March 2015.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-01 (work in progress), March 2015.

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

[OSCAR] Vucinic, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things", CoRR vol. abs/1404.7799, 2014.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

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

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

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

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

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

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

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

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

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

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

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI

October 19, 2015

Delegated CoAP Authentication and Authorization Framework (DCAF)
draft-gerdes-ace-dcaf-authorize-04

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

Copyright Notice

Table of Contents

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.ietf-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

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

Resource (R): A CoAP resource.

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.

Client Authorization Information (CAI): A data structure that describes the C’s permissions for S according to CAM, e.g., which actions C is allowed to perform on an R of S.

Server Authorization Information (SAI): A data structure that describes C’s permissions for S according to SAM, e.g., which actions C is allowed to perform on an R of S.

Verifier: The secret (e.g. a 128-bit PSK) shared between C and S. It enables C to validate that it is communicating with a certain S and vice versa.

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 flat or unrestricted authorization (cf section 4 of [I-D.ietf-ace-actors]).

2. System Overview

Within the DCAF Architecture each Server (S) has a Server Authorization Manger (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:

- o AM must have enough stable storage (such as disk space) to store the necessary number of credentials (matching the number of Clients and Servers).

- o 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.

- o 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 transfer between C and CAM; and

3. transfer of ticket 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. A formal specification of the DCAF message format is given in Appendix A.

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:

- o The request has been received on an unprotected channel.
- o S has no valid access ticket for the sender of the request regarding the requested action on that resource.

- o 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.

An optional field A lists the different content formats that are supported by S.

The message MAY also contain a timestamp generated by S.

Figure 2 shows an example for an 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,
 A: [ TBD1, ct_cose_msg ] }
```

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 content formats accepted by S are TBD1 (identifying 'application/dcaf+cbor' as defined in this document), and 'application/cose+cbor' defined in [I-D.ietf-cose-msg].

Editorial note: ct_cose_msg is to be replaced with the numeric value assigned for 'application/cose+cbor'.

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.

```
a2                                  # map(2)
    00                              # unsigned(0) (=SAM)
    78 21                           # text(33)
       636f6170733a2f2f73616d2e6578
       616d706c652e636f6d2f617574686f72
       697a65             # "coaps://sam.example.com/authorize"
    05                              # unsigned(5) (=TS)
    1a 00029259                     # unsigned(168537)
    0a                              # unsigned(10) (=A)
    82                              # array(2)
       19 03e6                      # unsigned(998) (=dcaf+cbor)
       19 03e7                      # unsigned(999) (=cose+cbor)
```

Figure 3: SAM Information Payload Example encoded in CBOR

3.3.1. Piggybacked Protected Content

For some use cases (such as sleepy nodes) it might be necessary to store sensor data on a server that might not belong to the same security domain. A client can retrieve the data from that server.

To be able to achieve the security objectives of the principles the data must be protected properly.

The server that hosts the stored data may respond to GET requests for this particular resource with a SAM Information message that contains the protected data as piggybacked content. As the server may frequently publish updates to the stored data, the URI of the authorization manager responsible for the protected data MAY be omitted and must be retrieved from a resource directory.

Once a requesting client has received the SAM Information Message with piggybacked content, it needs to request authorization for accessing the protected data. To do so, it constructs an Access Request as defined in Section 3.4. If access to the protected data is granted, the requesting client will be provided with cryptographic material to verify the integrity and authenticity of the piggybacked content and decrypt the protected data in case it is encrypted.

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 all requested actions are forbidden, CAM MUST send a 4.03 response.

If no authorization policies were specified or some or all of the requested actions are allowed according to the authorization policies, CAM either returns a cached response or attempts to create a Ticket Request message. The Ticket Request message MAY contain all actions requested by C since CAM will add CAI in the Ticket Transfer Message if COP specified authorization policies (see Section 3.7).

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.

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 "SAM" field that is specified in the Access Request message payload (for example, if the Access Request contained ’SAM: "coaps://sam.example.com/authz"’, the destination of the Ticket Request message is sam.example.com).

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.

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.

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 and the Client Information. Face contains the SAI and the Session Key Generation Method. The CI at this point only consists of the Verifier.

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.

```
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'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.

```
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 the ticket data 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 COP defined authorization policies that concern the requested actions, CAM MUST add Authorization Information for C (CAI) to the CI that reflect those policies. Since C and CAM use a DTLS channel for communication, the autorization information does not need to be encrypted.

CAM includes the Face and the CI containing the verifier sent by SAM in the Ticket Transfer message. However, CAM MUST NOT include additional information SAM provided in CI. In particular, CAM MUST NOT include any CAI information provided by SAM, since CAI represents COP’s authorization policies that MUST NOT be 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'f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9'
  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

When C receives a Ticket Transfer message, it checks if the payload contains a face and a Client Information. With this 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 TS 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 [RFC7641]) 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. If C repeatedly gets SAM Information messages as response to its requests, it SHOULD NOT send new Access Requests to CAM.

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.

```
    CAM                    C                    S                   SAM
     | <== DTLS chan. ==> | <== DTLS chan. ==> | <== DTLS chan. ==> |
     |                    |                    |                    |
     |                    | [Unauth. R. Req->] |                    |
     |                    |[<- 4.0x+SAM Info.] |                    |
     |                    |                    |                    |
     | <-- Access Req.    |                    |                    |
     |                    |                    |                    |
     | <==== TLS/DTLS channel (CAM/SAM Mutual Authentication) ====> |
     |                    |                    |                    |
     | Ticket Request   ----------------------------------------> |
     |                    |                    |                    |
     | <----------------------------------------    Ticket Grant  |
     |                    |                    |                    |
     | Ticket Transf. --> |                    |                    |
     |                    |                    |                    |
     |                    | <== Update SAI ==> |                    |
```

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 (CI). SAM generates the ticket Face for S and the verifier that corresponds to the ticket Face for C. The verifier is included in the CI.

The Ticket is transmitted over CAM to C. C keeps the CI and sends the Face to S. CAM can add Client authorization information (CAI) for C to the CI if necessary.

S uses the information in the ticket Face to validate that it was generated by SAM and to authenticate and authorize the client. No additional information about the Client is needed, S keeps the Ticket Face as long as it is valid.

C uses the verifier to authenticate S. If CAM specified CAI, the client uses it to authorize the server.

The ticket is not required to contain a client or a server identifier. The ticket Face MAY contain an SAI identifier for revocation. The CI MAY contain a CAI identifier for revocation.

4.1. Face

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

- o SAM Authorization Information (SAI)
- o A nonce

Optionally, Face MAY also contain:

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

- o A SAI identifier (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. 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).

Face MUST contain a nonce to verify that the contained information is fresh. As constrained devices may not have a clock, nonces 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 for the nonce. In this case, SAM and S MUST use a time synchronization mechanism to make sure that S interprets the timestamp correctly.

Face MAY contain an SAI identifier that uniquely identifies the SAI for S and SAM and can be used for revocation.

Face MAY be encrypted. If Face contains a DTLS PSK, the whole content of Face MUST be encrypted.

The ticket Face does not need to contain a client identifier.

4.2. Client Information

The CI part of the ticket is generated for C. It contains

- o The Verifier generated by SAM

CI MAY additionally contain:

- o CAI generated by CAM
- o A nonce generated by CAM
- o A lifetime generated by CAM
- o A SAI identifier generated by CAM

CI MUST contain the verifier, i.e. the DTLS PSK for C. The Verifier MUST NOT be transmitted over unprotected channels.

Additionally, CI MAY contain CAI to provide the COP's authorization policies to C. If the CI contains CAI, CAM MUST add a nonce that enables C to validate that the information is fresh. CAM MAY use a timestamp as the nonce (see Section 4.1). CAM SHOULD add a lifetime to CI to limit the lifetime of the CAI. CAM MAY additionally add a CAI identifier to CI for revocating the CAI. The CAI identifier MUST uniquely identify the CAI for C and CAM.

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.

The SAI in the ticket Face and the CAI in the CI need to be protected separately. CAM decides about the validity of the CAI while SAM is in charge of the validity of SAI. To be able to revoke the CAI, CAM SHOULD include a CAI identifier in the CI. SAM SHOULD include a SAI identifier in FACE to be able to revocate the SAI.

4.4. Lifetime

SAI and CAI MAY each have lifetime. SAM is responsible for defining the SAI lifetime, CAM is responsible for the CAI lifetime. If SAM sets a lifetime for SAI, 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.

4.4.1. 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.

Likewise, CAM MAY revoke tickets by sending a ticket revocation message to C. If C receives a CAI revocation message, it MUST end the DTLS connection to S and MUST NOT send any further requests to S.

If CAI revocation messages are used, C MUST check regularly if CAM is still available. If C cannot contact CAM, it MUST end all DTLS connections and MUST NOT send any more requests to S.

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.

Encoded Value	Key
0	SAM
1	SAI
2	CAI
3	E
4	K
5	TS
6	L
7	G
8	F
9	V
10	A
11	D
12	N

Table 1: DCAF field identifiers encoded in CBOR

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].

A: Accepted content formats. An array of numeric content formats from the CoAP Content-Formats registry (c.f. Section 12.3 of [RFC7252].

D: Protected Data. A binary string containing data that may be encrypted.

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. A 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. When in included in a ticket face, the contents of the L parameter denote the lifetime of the ticket. In combination with the protected data field D, this parameter denotes the lifetime of the protected data. 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.

N: Nonce. An initialization vector used in combination with piggybacked protected content.

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.

Encoded Value	Mnemonic	Support
0	hmac_sha256	mandatory
1	hmac_sha384	optional
2	hmac_sha512	optional

Table 2: CBOR encoding for DTLS PSK Key Generation Methods

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).

```
{
  F: {
       SAI: [ "/s/tempC", 1 ],
       TS: 2938749,
       L:  3600,
       G: hmac_sha256
     },
  V: h'48ae5a81b87241d81618f56cab0b65ec
       441202f81faabbe10075b20cb57fa939'
}
```

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:

- o SAM generates the DTLS PSK using an algorithm of its choice

- o 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(SAM,S) it shares with S. How SAM and S exchange K(SAM,S) is not in the scope of this document. SAM and S MAY use their preshared key as K(SAM,S).

- 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 S MUST decrypt Face using K(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

The constrained devices may be too constrained to manage complex trust relationships. Thus, DCAF does not require the constrained devices to perform complex tasks such as identifying a formerly unknown party. Each constrained device has a trust relationship with its respective AM. These less constrained devices are able to perform the more complex security tasks and can establish security associations with formerly unknown parties. The AMs hand down these security associations to their respective constrained device. The constrained devices require the help of their AMs for authentication and authorization.

C has a trust relationship with CAM: C trusts CAM to act in behalf of COP. S has a trust relationship with SAM: S trusts SAM to act in behalf of ROP. CAM trusts C to handle the data according to the CAI. SAM trusts S to protect resources according to the SAI. How the trust relationships between AMs and their respective constrained devices are established, is not in the scope of this document. It may be achieved by using a bootstrapping mechanism similar to [bergmann12] or by the means introduced in [I-D.gerdes-ace-a2a].

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

1. SAM and CAM have means to mutually authenticate each other (e.g., they might have a certificate of the other party or a PKI in which it is included)

2. If SAM requires information about the client from SAM, e.g. if SAM only wans to authorize certain types of devices, 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 a specific C.

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

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

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

```
CAM <------------------> SAM

/|\                      /|\
 |                        |
\|/                      \|/

 C ....................  S
```

9. Listing Authorization Manager Information in a Resource Directory

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

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

C --> CAM
POST client-authorize [Mid=1235,Token="tok"]
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 4]
}

CAM --> SAM [Mid=23146]
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", 4]
}

CAM <-- SAM
2.05 Content  [Mid=23146]
Content-Format: application/dcaf+cbor
{ F: {
       SAI: ["a/switch2941", 5],
       TS: 0("2013-07-04T20:17:38.002"),
       G: hmac_sha256
     },
  V: h'7ba4d9e287c8b69dd52fd3498fb8d26d
       9503611917b014ee6ec2a570d857987a'
}

C <-- CAM
2.05 Content  [Mid=1235,Token="tok"]
Content-Format: application/dcaf+cbor
{ F: {
       SAI: ["a/switch2941", 5],
       TS: 0("2013-07-04T20:17:38.002"),
       G: hmac_sha256
     },
  V: h'7ba4d9e287c8b69dd52fd3498fb8d26d
       9503611917b014ee6ec2a570d857987a'
```

```
}

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.

```
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'c7b5774f2ddcbd548f4ad74b30a1b2e5
       b6b04e66a9995edd2545e5a06216c53d'
}
```

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", 3]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor
{ F: {
       TS: 0("2013-07-16T10:15:43.663"),
       G: hmac_sha256
      },
  V: h'4f7b0e7fdcc498fb2ece648bf6bdf736
       61a6067e51278a0078e5b8217147ea06'
}
```

11. Specific Usage Scenarios

The general DCAF architure 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.-->] |                     |
  |                     |                     |
  |  [<-- SAM Info.]    |                     |
  |                     |                     |
  | <==== 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.

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.

```
     C                 CAM+SAM                 S
     | <== DTLS chan. ==> | <== DTLS chan. ==> |
     |                   |                     |
     | [Resource Req.----------------------->] |
     |                   |                     |
     | [<-------------------- SAM Information] |
     |                   |                     |
     | Access Request --> |                    |
     |                   |                     |
     | <-- Ticket Transf. |                    |
     |                   |                     |
     | <===========  DTLS channel ===========> |
     |                   |                     |
     | Authorized Resource Request ----------> |
```

Figure 10: Combined Client Authorization Manager and Server Authorization Manager

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.)

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

Figure 11: Combined Server Authorization Manager and Server

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:

Encoded Value	Name	Reference
0b000_00000	hmac_sha256	[RFC-XXXX]
0b000_00001	hmac_sha384	[RFC-XXXX]
0b000_00010	hmac_sha512	[RFC-XXXX]

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. Acknowledgements

The authors would like to thank Renzo Navas for his valuable input and feedback.

15. References

15.1. Normative References

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

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

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

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

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

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

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

15.2. Informative References

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-03 (work in progress), July 2015.

[I-D.gerdes-ace-a2a]
Gerdes, S., "Managing the Authorization to Authorize in the Lifecycle of a Constrained Device", draft-gerdes-ace-a2a-01 (work in progress), September 2015.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-07 (work in progress), October 2015.

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.

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

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 2015.

[I-D.schmertmann-dice-codtls]
Schmertmann, L., Hartke, K., and C. Bormann, "CoDTLS: DTLS handshakes over CoAP", draft-schmertmann-dice-codtls-01 (work in progress), August 2014.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <http://www.rfc-editor.org/info/rfc6655>.

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[bergmann12] Bergmann, O., Gerdes, S., Schaefer, S., Junge, F., and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network", IEEE Wireless Communications and Networking Conference Workshops (WCNCW), April 2012.

Appendix A. CDDL Specification

This appendix shows a formal specification of the DCAF messaging format using the CBOR data definition language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl]:

```
dcaf-msg = sam-information-msg
         / access-request-msg
         / ticket-transfer-msg
         / ticket-grant-msg

sam-information-msg = { sam, ? full-timestamp, ? accepted-formats,
                        ? piggybacked }

access-request-msg = { sam, sam-ai, full-timestamp }

ticket-transfer-msg = { face-or-encrypted, verifier }
face-or-encrypted = ( face | encrypted-face )
face = ( F => { sam-ai, limited-timestamp, lifetime, psk-gen } )
verifier = ( V => shared-secret )
shared-secret = bstr
F   = 8
V   = 9

encrypted-face = ( E => bstr, K => tstr )
E   = 3
K   = 4
```

```
ticket-grant-msg    = { face-or-encrypted, verifier, ? client-info }
client-info = ( cam-ai, full-timestamp, lifetime)

sam = (SAM => abs-uri)
SAM = 0
abs-uri = tstr ; .regexp "______"

sam-ai = ( SAI => [* auth-info])
SAI = 1
auth-info = ( uri : tstr, mask : 0..15 )

cam-ai = ( CAI => [* auth-info])
CAI = 2

full-timestamp = ( TS => date)
TS  = 5
date = tdate / localdate
localdate = uint
limited-timestamp = ( TS => localdate)

accepted-formats = ( A => [+ content-format] )
content-format = uint ; valid entry from CoAP content format registry
A=10

piggybacked = ( data, lifetime, nonce )
data = ( D => bstr )
none = ( N => bstr )
lifetime = ( L => period)
period = uint ; in seconds
L   = 6
D   = 11
N   = 12

psk-gen = ( G => mac-algorithm)
G   = 7
mac-algorithm = &( hmac-sha256: 0, hmac-sha384: 1, hmac-sha512: 2 )
```

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

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2016

L. Seitz, Ed.
SICS Swedish ICT AB
S. Gerdes, Ed.
Universitaet Bremen TZI
G. Selander
Ericsson
M. Mani
Itron
S. Kumar
Philips Research
October 23, 2015

Use Cases for Authentication and Authorization in Constrained Environments
draft-ietf-ace-usecases-10

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 includes a collection of representative use cases for 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

Copyright Notice

Table of Contents

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, deploying common security protocols can sometimes be difficult because of device or network limitations. Regardless, adequate security mechanisms are required to protect these constrained devices, which are expected to be integrated in all aspects of everyday life, from attackers wishing to gain control over the device's data or functions.

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].

2. Use Cases

This section includes the use cases; each use case first presents a general description of the application environment, than one or more specific use cases, and finally a summary of the authorization-related problems to be solved. The document aims at listing the relevant authorization problems and not to provide an exhaustive list. It might not be possible to address all of the listed problems with a single solution; There might be conflicting goals within or among some requirements.

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. Sensor systems make it possible to continuously track and transmit characteristics such as temperature, humidity and gas content while goods are transported and stored.

Sensors in this scenario have to be associated to the appropriate pallet of the respective container. Sensors as well as the goods belong to specific customers.

While in transit goods often pass stops where they are transloaded to other means of transportation, e.g. from ship transport to road transport.

Perishable goods need to be stored at constant temperature and with proper ventilation. Real-time information on the state of the goods is needed by both the transporter and the vendor. Transporters want to prioritize good that will expire soon. Vendors want to react when goods are spoiled to continue to fulfill delivery obligations.

The Intelligent Container (http://www.intelligentcontainer.com) is an example project that explores solutions to continuously monitor perishable goods.

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 (U1.2). Additionally, the sensor values are used to control the climate within the cargo containers (U1.1, U1.5, U1.7). 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 (U1.2, U1.3). 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 (U1.1).

The personnel that transloads the goods must be able to locate the goods meant for a specific customer (U1.1, U1.6, U1.7). 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 (U1.4). Thus, the transloading personnel is only allowed to access logistic information (U1.1). Moreover, the transloading personnel is only allowed to access the data for the time of the transloading (U1.8).

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 (U1.9). The sensors in the banana boxes cannot always reach the Internet during the journey (U1.10). Sensors may need to use relay stations owned by the transport company to connect to endpoints in the Internet.

In the ripening facility bananas are stored until they are ready to be sold. 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 (U1.2, U1.8).

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 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 and authenticity 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 and authenticity 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 and the confidentiality of location data, e.g., to protect them from targeted attacks from competitors.

- U1.5 The fruit vendor may need different protection for several different types of data on the same endpoint, e.g., sensor data and the data used for logistics.

- U1.6 The fruit vendor and the transloading personnel require the authenticity and integrity of the data that is used to locate the goods, in order to ensure that the goods are correctly treated and delivered.

- U1.7 The container owner and the fruit vendor may not be present at the time of access and cannot manually intervene in the authorization process.

- U1.8 The fruit vendor, container owner and transloading company want to grant temporary access permissions to a party, in order to avoid giving permanent access to parties that are no longer involved in processing the bananas.

- U1.9 The fruit vendor, container owner and transloading company want their security objectives to be achieved, even if the messages between the endpoints need to be forwarded over multiple hops.

- U1.10 The constrained devices might not always be able to reach the Internet but still need to enact the authorization policies of their principals.

- o U1.11 Fruit vendors and container owners want to be able to revoke authorization on a malfunctioning sensor.

2.2. Home Automation

One application of the Internet of Things is home automation systems. Such a system can connect household devices that control, for example heating, ventilation, lighting, home entertainment, and home security to the Internet making them remotely accessible and manageable.

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 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 (U2.5).

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 (U2.5).

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 (U2.8).

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 (U2.7). She provides the necessary configurations for that (U2.9, U2.10).

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 (U2.1, U2.6). Then Alice sets temporary permissions that allow them to open the door, and shut down the alarm (U2.2). 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 (U2.3, U2.4).

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 (U2.5, U2.9). The permissions Alice issued to her parents only allow limited access to the house (e.g. opening the door, turning on the lights). Certain other functions, such as checking the footage from the surveillance cameras is not accessible to them (U2.3).

Alice and Bob also issue similarly restricted permissions to e.g. cleaners, repairmen or their nanny (U2.3).

2.2.4. Selling the house

Alice and Bob have to move because Alice is starting a new job. They therefore decide to sell the house, and transfer control of all automated services to the new owners (U2.11). Before doing that they want to erase privacy relevant data from the logs of the automated systems, while the new owner is interested to keep some historic data e.g. pertaining to the behavior of the heating system (U2.12). At the time of transfer of the house, the new owners also wants make sure that permissions issued by the previous owners to access the house or connected devices (in the case where device management may have separate permissions from house access) are no longer valid (U2.13).

2.2.5. 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.

- o U2.3 A home owner wants to apply different access rights for different users (including other inhabitants).

- o U2.4 The home owners want to grant access permissions to a someone during a specified time frame.

- o U2.5 The smart home devices need to be able to securely communicate with different control devices (e.g. wall-mounted touch panels, smartphones, electronic key fobs, device gateways).

- o U2.6 The home owner wants to be able to configure authorization policies remotely.

- o U2.7 Authorized Users want to be able to obtain access with little effort.

- o 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.

- o 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.

- o U2.10 Home Owners want their devices to seamlessly (and in some cases even unnoticeably) fulfill their purpose. Therefore the authorization administration effort needs to be kept at a minimum.

- o U2.11 Home Owners want to be able to transfer ownership of their automated systems when they sell the house.

- o U2.12 Home Owners want to be able to sanitize the logs of the automated systems, when transferring ownership, without deleting important operational data.

- o U2.13 When a transfer of ownership occurs, the new owner wants to make sure that access rights created by the previous owner are no longer valid.

2.3. Personal Health Monitoring

Personal health monitoring devices, i.e. eHealth devices, are typically battery driven and located physically on or in the user to monitor some bodily function, such as temperature, blood pressure, or pulse rate. These devices typically connect to the Internet through an intermediary base-station, using wireless technologies and through

this connection they report the monitored data to some entity, which may either be the user, or a medical caregiver.

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.

There is a plethora of wearable health monitoring technology and 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).

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 location (U3.7). In case of a cardiac arrest it automatically sends an alarm to an emergency service, transmitting John’s current location (U3.1). Either the device has long range connectivity itself (e.g. via GSM) or it uses some intermediary, nearby device (e.g. John’s smartphone) to transmit such an alarm. To ensure Johns safety, the device is expected to be in constant operation (U3.3, U3.6).

The device includes an authentication mechanism, in order to prevent other persons who get physical access to it from acting as the owner and altering the access control and security settings (U3.8).

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 (U3.2).

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 (U3.8).

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 (U3.5). 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 (U3.4).

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 Battery changes are very inconvenient and sometimes impractical, so battery life impacts of the authorization mechanisms need to be minimized.

- U3.4 Devices are often used with default access control settings which might threaten the security objectives of the device’s users.

- 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, especially on the constrained devices.

- U3.7 The device provides a service that can be fatal for the wearer if it fails. Accordingly, the wearer wants the device to

have a high degree of resistance against attacks that may cause the device to fail to operate partially or completely.

- o U3.8 The wearer of an eHealth device requires the integrity and confidentiality of the data measured by the device.

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. These components are being increasingly managed centrally in a Building and Lighting Management System (BLMS) by a facility manager.

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 lighting 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 facility management company.

2.4.1. Device Lifecycle

2.4.1.1. Installation and Commissioning

Installation of the building automation components often start even before the construction work is completed. Lighting is one of the first components to be installed in new buildings. A lighting plan created by a lighting designer provides the necessary information related to the kind of lighting devices (luminaires, sensors and switches) to be installed along with their expected behavior. The physical installation of the correct lighting devices at the right locations are done by electricians based on the lighting plan. They ensure that the electrical wiring is performed according to local regulations and lighting devices which may be from multiple manufacturers are connected to the electrical power supply properly. After the installation, lighting can be used in a default out-of-box mode for e.g. at full brightness when powered on. After this step (or in parallel in a different section of the building), a lighting commissioner adds the devices to the building domain (U4.1) and performs the proper configuration of the lights as prescribed in the

lighting plan. This involves for example grouping to ensure that light points react together, more or less synchronously (U4.8) and defining lighting scenes for particular areas of the building. The commissioning is often done in phases, either by one or more commissioners, on different floors. The building lighting network at this stage may be in different network islands with no connectivity between them due to lack of the IT infrastructure.

After this, other building components like HVAC and security systems are similarly installed by electricians and later commissioned by their respective domain professionals. Similar configurations related to grouping (U4.8) are required to ensure for e.g. HVAC equipment are controlled by the closest temperature sensor.

For the building IT systems, the Ethernet wiring is initially laid out in the building according to the IT plan. The IT network is commissioned often after the construction is completed to avoid any damage to sensitive networking and computing equipment. The commissioning is performed by an IT engineer with additional switches (wired and/or wireless), IP routers and computing devices. Direct Internet connectivity for all installed/commissioned devices in the building is only available at this point. The BLMS that monitors and controls the various building automation components are only connected to the field devices at this stage. The different network islands (for lighting and HVAC) are also joined together without any further involvement of domain specialist such as lighting or HVAC commissioners.

2.4.1.2. Operational

The building automation systems is now finally ready and the operational access is transferred to the facility management company of the building (U4.2). The facility manager is responsible for monitoring and ensuring that the building automation systems meets the needs of the building occupants. If changes are needed, the facility management company hires an external installation and commissioning company to perform the changes.

Different parts of the building are rented out to different companies for office space.
The tenants are provided access to use the automated HVAC, lighting and physical access control systems deployed. The safety of the occupants are also managed using automated systems, such as a fire alarm system, which is triggered by several smoke detectors which are spread out across the building.

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 rooms 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 and associated building automation components with company A (U4.2, U4.9).

2.4.1.3. Maintenance

Company A’s staff are annoyed that the lighting switches off too often in their rooms if they work silently in front of their computer. Company A notifies the the facility manager of the building to increase the delay before lights switch off. The facility manager can either configure the new values directly in the BLMS or if additional changes are needed on the field devices, hires a commissioning Company C to perform the needed changes (U4.4).

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

At some point the facility management company wants to update the firmware of lighting devices in order to eliminate software bugs. Before accepting the new firmware, each device checks the authorization of the facility management company to perform this update (U4.13).

A network diagnostic tool of the BLMS detects that a luminaire in one of the Company A’s office room is no longer connected to the network. The BLMS alerts the facility manager to replace the luminaire. The facility manager replaces the old broken luminaire and informs the BLMS of the identity (for e.g. MAC address) of the newly added device. The BLMS then authorizes the new device onto the system and transfers seamlessly all the permissions of the previous broken device to the replacement device (U4.12).

2.4.1.4. Recommissioning

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 (U4.1).

Company C gets the necessary authorization from the facility management company to interact with the existing BLMS (U4.4). 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 (U4.5). Before removing existing devices, all security and configuration material that belongs to the domain are deleted and the devices are set back to factory state (U4.3). This ensures that these devices may be reused at other installations or in other parts of the same building without affecting future operations. 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 (U4.7). 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 (U4.8). 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 (U4.5).

2.4.1.5. 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 (U4.4).

Company C again gets the necessary authorization from the facility management company to interact with the BLMS. The commissioner now deletes any rules that allowed handheld controllers authorization to control the lighting (U4.3, U4.6). Additionally the commissioner instructs the BLMS to push these new rules to prevent cached rules at the end devices from being used. Any cryptographic key material belonging to the site in the handheld controllers are also removed and they are set to the factory state (U4.3).

2.4.2. Public Safety

The fire department requires that as part of the building safety code, that the building have sensors that sense the level of smoke, heat, etc., when a fire breaks out. These sensors report metrics which are then used by a back-end server to map safe areas and unsafe areas within a building and also possibly the structural integrity of the building before fire-fighters may enter it. Sensors may also be used to track where human/animal activity is within the building. This will allow people stuck within the building to be guided to safer areas and suggest possible actions that they may take (e.g. using a client application on their phones, or loudspeaker directions) in order to bring them to safety. In certain cases, other organizations such as the Police, Ambulance, and federal organizations are also involved and therefore the coordination of tasks between the various entities have to be carried out using efficient messaging and authorization mechanisms.

2.4.2.1. A fire breaks out

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 group communication scheme) to alert the staff of both companies (U4.8). 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 (using group communication) to switch off all the air conditioning (U4.10).

The fire department is notified of the fire automatically and arrives within a short time. They automatically get access to all parts of the building according to an emergency authorization policy (U4.4, U4.5). 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 (U4.4, U4.11).

2.4.3. Authorization Problems Summary

- o U4.1 During commissioning, the building owner or the companies add new devices to their administrative domain. Access control should then apply to these devices seamlessly.

- o U4.2 During a handover, the building owner or the companies integrate devices that formerly belonged to a different administrative domain to their own administrative domain. Access control of the old domain should then cease to apply, with access control of the new domain taking over.

- o U4.3 During decommissioning, the building owner or the companies remove devices from their administrative domain. Access control should cease to apply to these devices and relevant credentials need to be erased from the devices.

- o U4.4 The building owner and the companies want to be able to delegate specific access rights for their devices to others.

- o 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 group communication scheme.

- 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.

- o U4.10 The authorization mechanisms must be able to cope with extremely time-sensitive operations which have to be carried out in a quick manner.

- o U4.11 The building owner and the public safety authorities want to be able to perform data origin authentication on messages sent and received by some of the systems in the building.

- o U4.12 The building owner should be allowed to replace an existing device with a new device providing the same functionality within their administrative domain. Access control from the replaced device should then apply to these new devices seamlessly.

- o U4.13 When software on a device is updated, this update needs to be authenticated and authorized.

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.

Various stakeholders have a claim on the metering data. Utility companies need the data for accounting, the metering equipment may be operated by a third party Service Operator who needs to maintain it, and the equipment is installed in the premises of the consumers, measuring their consumption, which entails privacy questions.

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, they can also be used to shut off the water if the bills are not paid (U5.1, U5.3). The meters do so by sending and receiving data to and from a base station (U5.2). 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 (U5.4, U5.6, U5.8).

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 (U5.9).

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 (U5.2, U5.7) and reduce costs by supporting new tariff models from utility companies, and more accurate and timely billing. However the end-consumers do not want unauthorized persons to gain access to this data. Furthermore, the fine-grained measurement of consumption data may induce privacy concerns, since it may allow others to create behavioral profiles (U5.5, U5.10).

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 (U5.9). For reasons of efficiency and cost, end-to-end connectivity is not always feasible, so metering data is stored and aggregated in various intermediate devices before being forwarded to the utility company, and in turn accessed by the MDM. The intermediate devices may be operated by a third party service operator on behalf of the utility company (U5.7). One responsibility of the service operator is to make sure that meter readings are performed and delivered in a regular, timely manner. 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

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

- o U5.2 The utility company wants to control which entities are allowed to send data to, and read data from their endpoints.

- o U5.3 The utility company wants to ensure the integrity of the data stored on their endpoints.

- o U5.4 The utility company wants to protect such data transfers to and from their endpoints.

- o U5.5 Consumers want to access their own usage information and also prevent unauthorized access by others.

- o U5.6 The devices may have intermittent Internet connectivity but still need to enact the authorization policies of their principals.

- o U5.7 Neither the service operator nor the utility company are always present at the time of access and cannot manually intervene in the authorization process.

- o U5.8 When authorization policies are updated it is impossible, or at least very inefficient to contact all affected endpoints directly.

- o U5.9 Authorization and authentication must work even if messages between endpoints are stored and forwarded over multiple nodes.

- o U5.10 Consumers may not want the Service Operator, the Utility company or others to have access to a fine-grained level of consumption data that allows the creation of behavioral profiles.

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 functions 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.

Continuously monitoring allows authorized users to create behavioral or movement profiles, which corresponds on the devices intended use, and unauthorized access to the collected data would allow an attacker to create the same 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 a 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 (U6.4). 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 (U6.1, U6.2). 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 (U6.3). Jody wants the device to report the data back to her fitness account while she borrows it, so she allows it to access her account temporarily.

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

2.6.2. Authorization Problems Summary

- o U6.1 Sports equipment owners want to be able to grant access rights dynamically when needed.

- o U6.2 Sports equipment owners want the configuration of access rights to work with very little effort.

- o U6.3 Sports equipment owners want to be able to pre-configure 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.

- o U6.4 Sports equipment owners want 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 these kind of systems are, especially when connected to the Internet [Karnouskos11]. The severity of these vulnerabilities are exacerbated by the fact that many ICS are used to control critical public infrastructure, such as nuclear 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 (U7.5).

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 (U7.4).

Some of 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 (U7.3).

The main interest of the operator is to ensure the integrity of control messages and sensor readings (U7.1). Access in some cases needs to be restricted, e.g. the operator wants wireless actuators only to accept commands by authorized control units (U7.2).

The owner of the platform also wants to collect auditing information for liability reasons (U7.1).

Different levels of access apply e.g. for regular operators, vs. maintenance technician, vs. auditors of the platform (U7.6)

2.7.2. Authorization Problems Summary

- o U7.1 The operator of the platform wants to ensure the integrity and confidentiality of sensor and actuator data.

- o U7.2 The operator wants to ensure that data coming from sensors and commands sent to actuators are authentic.

- o U7.3 Some devices do not have direct Internet connection, but still need to implement current authorization policies.

- o U7.4 Devices need to authenticate the controlling units, especially those using a wireless connection.

- o U7.5 The execution of unauthorized commands or the failure to execute an authorized command in an ICS can lead to significant financial damage, and threaten the availability of critical infrastructure services. Accordingly, the operator wants a authentication and authorization mechanisms that provide a very high level of security.

- o U7.6 Different users should have different levels of access to the control system (e.g. operator vs. auditor).

3. Security Considerations

As the use cases listed in this document demonstrate, constrained devices are used in various environments. These devices are small and inexpensive and this makes it easy to integrate them into many aspects of everyday life. With access to vast amounts of valuable data and possibly control of important functions these devices need to be protected from unauthorized access. Protecting seemingly innocuous data and functions will lessen the possible effects of aggregation; attackers collecting data or functions from several sources can gain insights or a level of control not immediately obvious from each of these sources on its own.

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 gains 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. Solution Designers should consider this in the design of their security solution and provide for protection against this type of attack. In particular, messages containing sensitive data that are send over unprotected channels should be encrypted if possible.

Attacks aimed at altering data in transit (e.g. to perpetrate fraud) are a problem that is addressed in many web security protocols such as TLS or IPSec. Developers need to consider this type of attacks, and make sure that the protection measures they implement are adapted to the constrained environment.

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 this document, but should be kept in mind by developers of authorization solutions.

Denial of service (DoS) attacks threaten the availability of services a device provides and constrained devices are especially vulnerable to these types of attacks because of their limitations. Attackers can illicit a temporary or, if the battery is drained, permanent failure in a service simply by repeatedly flooding the device with connection attempts; for some services (see section Section 2.3), availability is especially important. Solution designers must be particularly careful to consider the following limitations in every part of the authorization solution:

- o Battery usage
- o Number of required message exchanges
- o Size of data that is transmitted (e.g. authentication and access control data)

- o Size of code required to run the protocols

- o Size of RAM memory and stack required to run the protocols

- o Resources blocked by partially completed exchanges (e.g. while one party is waiting for a transaction time to run out)

Solution developers also need to consider whether the session should be protected from information disclosure and tampering.

3.2. Configuration of Access Permissions

- o 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.

- o 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.

- o A device might host several resources where each resource has its own access control policy (all use cases).

- o The device that makes the policy decisions should be able to evaluate context-based permissions such as location or time of access (see 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. Authorization Considerations

- o Devices need to be enabled to enforce authorization policies without human intervention at the time of the access request (see Section 2.1, Section 2.2, Section 2.4, Section 2.5).

- o Authorization solutions need to consider that constrained devices might not have internet access at the time of the access request (see Section 2.1, Section 2.3, Section 2.5, Section 2.6).

- o It should be possible to update access control policies without manually re-provisioning individual devices (see Section 2.2, Section 2.3, Section 2.5, Section 2.6).

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

- o 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 Section 2.5).

- o It must be possible to dynamically revoke authorizations (see e.g. Section 2.4).

- o 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).

- o 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.

- o 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).

- o 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 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).

- o Software updates are an important operation for which correct authorization is crucial. Additionally authenticating the receiver of a software update is also important, for example to make sure that the update has been received by the intended device.

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

The constrained devices in focus of this document collect data from the physical world via sensors or affect their surrounding via actuators. The collected and processed data often can be associated with individuals. Since sensor data may be collected and distributed on a regular interval a significant amount of information about an individual can be collected and used as input to learning algorithms as part of big data analysis and used in an automated decision making process.

Offering privacy protection for individuals is important to guarantee that only authorized entities are allowed to access collected data and to trigger actions, to obtain consent prior to the sharing of data, and to deal with other privacy-related threats outlined in RFC 6973.

RFC 6973 was written as guidance for engineers designing technical solutions. For a short description about the deployment-related aspects of privacy and further references relevant for the Internet of Things sector please read Section 7 of RFC 7452.

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, Andreas Baeckman, Samuel Erdtman, Steve Moore, Thomas Hardjono, Kepeng Li, Jim Schaad, Prashant Jhingran, Kathleen Moriarty, and Sean Turner 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. Furthermore the authors thank Akbar Rahman, Chonggang Wang, Vinod Choyi, and Abhinav Somaraju who contributed to the building automation 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]
Jedermann, R., Poetsch, T., and C. LLoyd, "Communication techniques and challenges for wireless food quality monitoring", Philosophical Transactions of the Royal

Society A Mathematical, Physical and Engineering Sciences, May 2014.

[Karnouskos11]
Karnouskos, S., "Stuxnet Worm Impact on Industrial Cyber-Physical System Security", IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, pp. 4490-4494 , November 2011.

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

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

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

Authors' Addresses

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

Email: ludwig@sics.se

Stefanie Gerdes (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

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

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Mehdi Mani
Itron
52, rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: Mehdi.Mani@itron.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

H. Tschofenig
ARM Limited
E. Maler
Forgerock
E. Wahlstroem
S. Erdtman
Nexus Technology
March 9, 2015

Authentication and Authorization for Constrained Environments Using OAuth and UMA

draft-maler-ace-oauth-uma-00.txt

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

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

Table of Contents

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.

```
                              +-------------+
                              |Authorization|
                              |Server (AS)  |\
                              +-------------+ \
                              ^        /   ^    \
                    Request   /        /     \     \     *Token
                    Access   / Access /       \     \  Introspection
                    Token   /  Token /          \     \
                           /        /            \     \
                          /        /              \      \
                         /        /                \      \
                        /        /                  \      \
          O            /        v                    \       v
         /|\          +-----------+                  +-----------+
          |    -----> |           |   Access Token   | Resource  |
         / \   <----- |  Client   |----------------->|  Server   |
      Resource        |           |<================>|   (RS)    |
       Owner          +-----------+ Application Data +-----------+

         *: indicates optional exchange.
```

Figure 1: OAuth Architecture.

```
    +--------+                                +---------------+
    |        |--(A)- Authorization Request ->|   Resource    |
    |        |                               |     Owner     |
    |        |<-(B)-- Authorization Grant ---|               |
    |        |                               +---------------+
    |        |
    |        |                               +---------------+
    |        |--(C)-- Authorization Grant -->| Authorization |
    | Client |                               |     Server    |
    |        |<-(D)----- Access Token -------|               |
    |        |                               +---------------+
    |        |                                  ^ |    *Token
    |        |                               (F)| |(G) Introspection
    |        |                                  | v
    |        |                               +---------------+
    |        |--(E)----- Access Token ------>|    Resource   |
    |        |                               |     Server    |
    |        |<-(H)--- Protected Resource ---|               |
    +--------+                               +---------------+
```

Figure 2: OAuth 2.0 Message Exchange.

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.

```
                                          +--------------+
                                          |   resource   |
            +---------manage (A)------------ |     owner    |
            |                             +--------------+
            |         Phase 1:                   |
            |         protect a               control (C)
            |         resource                   |
            v                                    v
      +------------+              +----------+--------------+
      |            |              |protection|              |
      |  resource  |              |   API    | authorization|
      |   server   |<-protect (B)--|  (needs  |    server    |
      |            |              |   PAT)   |              |
      +------------+              +----------+--------------+
      | protected  |                         | authorization|
      | resource   |                         |     API      |
      |(needs RPT) |                         |  (needs AAT) |
      +------------+                         +--------------+
            ^                                    |
            |         Phases 2 and 3:         authorize (D)
            |         get authorization,         |
            |         access a resource          v
            |                             +--------------+
            +---------access (E)-------------|    client    |
                                          +--------------+

                                          requesting party
```

Figure 3: OAuth++: The UMA Architecture.

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 similiar 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:

- o 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.

- o 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.

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

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

- o 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.

- o 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.

- o 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 resouce 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 smart phone 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 compay 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-limitable 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 protectoin 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 desireable 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]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-12 (work in progress), February 2015.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 2015.

[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 Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

[RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.

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

10.2. Informative References

[I-D.iab-smart-object-architecture]
Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", draft-iab-smart-object-architecture-06 (work in progress), October 2014.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-02 (work in progress), February 2015.

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

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

ACE Working Group
Internet-Draft
Intended Status: Informational
Expires: September 10, 2015

L. Seitz
SICS Swedish ICT
G. Selander
Ericsson

March 9, 2015

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

Copyright and License Notice

Table of Contents

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 polices 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

```
                    +---------------+
                    | Authorization |
                    |     Server    |
                    |               |
                    +---------------+
                   /                 \    Authorization
  Credentials,    /                   \    Information
       Keys      /                     \
                /                       \   Credentials,
               /                         \   Keys
              V                           V
     +--------+                           +-----------+
     | Client |                           | Resource  |
     |        |<---- Access procedure --->| Server    |
     |        |                           |           |
     +--------+                           +-----------+
```

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

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:

- o AS needs to transfer authorization information to RS.
- o The transferred authorization information needs to follow a defined format and encoding, which must be efficient for constrained devices, considering size of authorization information and parser complexity.
- o 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.
- o 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.
- o Finally, as is indicated in the previous bullet, for a particular authorization decision there may be different kinds of authorization information needed, and these pieces of information may be transferred to 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:

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

- o C may need to to authenticate AS to ensure that it gets security information related to the resources from the right source.

- o In some use cases RS needs to authenticate some property of C, in order to 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.

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:

- 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-coapmq]). 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.

3.4. Cryptographic Keys

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

- o 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.

- o 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.

- o 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:

- o RS hosting resources, and responding to access requests
- o C requesting access to resources
- o 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.
- o 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

- o C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover
  - unable to manage complex authorization policies
  - unable to manage a large number of secure connections
  - without user interface
  - without constant network connectivity
  - unable to precisely measure time
  - required to save on wireless communication due to high power consumption
- o AS is not a constrained device.
- o 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.

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

o 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.

o 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

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

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

4.4 Authorization

o 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.

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

o 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.

o Apart from authorization for access to a resource, authorization may also be required for access to information about a resource

(e.g. resource descriptions).

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

4.5 Authorization Information

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

4.6 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).

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

- o The solution shall work with existing infrastructure.

- o 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:

- o Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.

- o Denial-of-service making the sensor/actuator unable to perform its intended task correctly.

A number of attacks can be made with physical access to a device including probing attacks, timing attacks, power attacks, etc. However, with physical access to a sensor or actuator device it is possible to directly perform attacks equivalent of eavesdropping, manipulating data or denial of service. For example:

- o 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.

- o 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.

- o 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

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-02 (work in progress), February 2015.

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M. and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-12 (work in progress), February 2015.

[I-D.selander-ace-object-security] Selander G., Mattsson J., and L. Seitz, "Object Security for CoAP", draft-selander-ace-object-security-00 (work in progress), October 2014.

[I-D.koster-core-coapmq]
Koster, M., Keranen A., and J. Jimenez "Message Queueing in the Constrained Application Protocol (CoAP)", draft-koster-core-coapmq-00 (expired), July 2014

[I-D.gerdes-ace-actors]
Gerdes, S., "Actors in the ACE Architecture", draft-gerdes-ace-actors-02 (work in progress), October 2014.

[HUM14delegation] Hummen, R., Shafagh, H., Raza, S., Voigt, T., Wehrle, K., "Delegation-based Authentication and Authorization for the IP-based Internet of Things", 11th IEEE International Conference on Sensing, Communication, and Networking (SECON’14), June 30 - July 3, 2014.

[RFC2748] Durham, D., Ed., Boyle, J., Cohen, R., Herzog, S., Rajan, R., and A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000, <http://www.rfc-editor.org/info/rfc2748>.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000, <http://www.rfc-editor.org/info/rfc2904>.

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

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

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

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

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

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

Authors' Addresses

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

Goeran Selander
Ericsson
Farogatan 6

16480 Kista
SWEDEN
EMail: goran.selander@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 14, 2017

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
SICS Swedish ICT
October 11, 2016

Object Security of CoAP (OSCOAP)
draft-selander-ace-object-security-06

Abstract

This memo defines Object Security of CoAP (OSCOAP), a method for application layer protection of message exchanges with the Constrained Application Protocol (CoAP), using the CBOR Object Signing and Encryption (COSE) format. OSCOAP provides end-to-end encryption, integrity and replay protection to CoAP payload, options, and header fields, as well as a secure binding between CoAP request and response messages. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web application protocol, designed for constrained nodes and networks [RFC7228]. CoAP specifies the use of proxies for scalability and efficiency. At the same time CoAP references DTLS [RFC6347] for security. Proxy operations on CoAP messages require DTLS to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of the CoAP payload and metadata, in transit between client and server. The proxy can also inject, delete, or reorder packages without being protected or detected by DTLS.

This memo defines Object Security of CoAP (OSCOAP), a data object based security protocol, protecting CoAP message exchanges end-to-end, across intermediary nodes. An analysis of end-to-end security for CoAP messages through intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs], this specification addresses the forwarding case.

The solution provides an in-layer security protocol for CoAP which does not depend on underlying layers and is therefore favorable for providing security for "CoAP over foo", e.g. CoAP messages passing over both unreliable and reliable transport [I-D.ietf-core-coap-tcp-tls], CoAP over IEEE 802.15.4 IE [I-D.bormann-6lo-coap-802-15-ie].

OSCOAP builds on CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg], providing end-to-end encryption, integrity, and replay protection. The use of OSCOAP is signaled with the CoAP option Object-Security, also defined in this memo. The solution transforms an unprotected CoAP message into a protected CoAP message in the following way: the unprotected CoAP message is protected by including payload (if present), certain options, and header fields in a COSE object. The message fields that have been encrypted are removed from the message whereas the Object-Security option and the COSE object are added. We call the result the "protected" CoAP message. Thus OSCOAP is a security protocol based on the exchange of protected CoAP messages (see Figure 1).

```
          Client                                          Server
             |  request:                                     |
             |    GET example.com                            |
             |    [Header, Token, Options:{...,              |
             |     Object-Security:COSE object}]             |
             +----------------------------------------------->|
             |  response:                                    |
             |    2.05 (Content)                             |
             |    [Header, Token, Options:{...,              |
             |     Object-Security:-}, Payload:COSE object]  |
             |<-----------------------------------------------+
             |                                               |
```

Figure 1: Sketch of OSCOAP

OSCOAP provides protection of CoAP payload, certain options, and header fields, as well as a secure binding between CoAP request and response messages, and freshness of requests and responses. It may be used in extremely constrained settings, where DTLS cannot be supported. Alternatively, OSCOAP can be combined with DTLS, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node. Examples of the use of OSCOAP are given in Appendix B.

The message protection provided by OSCOAP can alternatively be applied only to the payload of individual messages. We call this object security of content (OSCON) and it is defined in Appendix C.

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.

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

Two different scopes of object security are defined:

- o OSCOAP = object security of CoAP, signaled with the Object-Security option.

- o OSCON = object security of content, signaled with Content Format/ Media Type set to application/oscon (defined in Appendix C).

2. The Object-Security Option

The Object-Security option indicates that OSCOAP is used to protect the CoAP message exchange. The protection is achieved by means of a COSE object included in the protected CoAP message, as detailed in Section 5.

The Object-Security option is critical, safe to forward, part of the cache key, and not repeatable. Figure 2 illustrates the structure of the Object-Security option.

A CoAP proxy SHOULD NOT cache a response to a request with an Object-Security option, since the response is only applicable to the original client’s request. The Object-Security option is included in the cache key for backward compatibility with proxies not recognizing the Object-Security option. The effect of this is that messages with the Object-Security option will never generate cache hits. To further prevent caching, a Max-Age option with value zero SHOULD be added to the protected CoAP responses.

```
+-----+---+---+---+---+-----------------+--------+--------+
| No. | C | U | N | R | Name            | Format | Length |
+-----+---+---+---+---+-----------------+--------+--------|
| TBD | x |   |   |   | Object-Security | opaque | 0-     |
+-----+---+---+---+---+-----------------+--------+--------+
     C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable
```

Figure 2: The Object-Security Option

The length of the Object-Security option depends on whether the unprotected message has payload, on the set of options that are included in the unprotected message, the length of the integrity tag, and the length of the information identifying the security context.

- o If the unprotected message has payload, then the COSE object is the payload of the protected message (see Section 6.2 and Section 6.4), and the Object-Security option has length zero. An endpoint receiving a CoAP message with payload, that also contains a non-empty Object-Security option SHALL treat it as malformed and reject it.

- o If the unprotected message does not have payload, then the COSE object is the value of the Object-Security option and the length of the Object-Security option is equal to the size of the COSE object. An endpoint receiving a CoAP message without payload,

that also contains an empty Object-Security option SHALL treat it as malformed and reject it.

More details about the message overhead caused by the Object-Security option is given in Appendix A.

3. The Security Context

OSCOAP uses COSE with an Authenticated Encryption with Additional Data (AEAD) algorithm. The specification requires that client and server establish a security context to apply to the COSE objects protecting the CoAP messages. In this section we define the security context, and also specify how to establish a security context in client and server based on common shared secret material and a key derivation function (KDF).

The EDHOC protocol [I-D.selander-ace-cose-ecdhe] enables the establishment of secret material with the property of forward secrecy, and negotiation of KDF and AEAD, it thus provides all necessary pre-requisite steps for using OSCOAP as defined here.

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCOAP. Each security context is identified by a Context Identifier. A Context Identifier that is no longer in use can be reassigned to a new security context.

For each endpoint, the security context is composed by a "Common Context", a "Sender Context" and a "Recipient Context". The Common Context includes common security material. The endpoint protects the messages sent using the Sender Context. The endpoint verifies the messages received using the Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Note that, because of that, the two security contexts identified by the same Context Identifiers in the two endpoints are not the same, but they are partly mirrored.

An example is shown in Figure 3.

```
                  .-Cid = Cid1-.            .-Cid = Cid1-.
                  | context:   |            | context:   |
                  |  Alg,      |            |  Alg,      |
                  |  Sender,   |            |  Recipient,|
                  |  Recipient |            |  Sender    |
                  '------------'            '------------'
                      Client                    Server
                         |                       |
   Retrieve context for  | request:              |
    target resource      |  [Token = Token1,     |
   Protect request with  |    Cid=Cid1, ...]     |
     Sender              +----------------------->| Retrieve context with
                         |                       |  Cid = Cid1
                         |                       | Verify request with
                         |                       |  Recipient
                         | response:             | Protect response with
                         |  [Token = Token1, ...]|  Sender
   Retrieve context with |<----------------------+
    Token = Token1       |                       |
   Verify request with   |                       |
    Recipient            |                       |
```

Figure 3: Retrieval and use of the Security Context

The Common Context structure contains the following parameters:

- o Context Identifier (Cid). Variable length byte string that identifies the security context. Its value is immutable once the security context is established.

- o Algorithm (Alg). Value that identifies the COSE AEAD algorithm to use for encryption. Its value is immutable once the security context is established.

- o Base Key (base_key). Byte string containing the key used to derive the security context Section 3.2.

The Sender Context structure contains the following parameters:

- o Sender ID. Variable length byte string identifying oneself. Its value is immutable once the security context is established.

- o Sender Key. Byte string containing the symmetric key to protect messages to send. Length is determined by Algorithm. Its value is immutable once the security context is established.

- o Sender IV. Byte string containing the fixed portion of IV (context IV in [I-D.ietf-cose-msg]) to protect messages to send.

Length is determined by Algorithm. Its value is immutable once the security context is established.

- o Sender Sequence Number. Non-negative integer enumerating the COSE objects that the endpoint sends, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.

The Recipient Context structure contains the following parameters:

- o Recipient ID. Variable length byte string identifying the endpoint messages are received from or sent to. Its value is immutable once the security context is established.

- o Recipient Key. Byte string containing the symmetric key to verify messages received. Length is determined by the Algorithm. Its value is immutable once the security context is established.

- o Recipient IV. Byte string containing the context IV to verify messages received. Length is determined by Algorithm. Its value is immutable once the security context is established.

- o Recipient Sequence Number. Non-negative integer enumerating the COSE objects received, associated to the Context Identifier. It is used for replay protection, and to generate unique IVs for the AEAD. Maximum value is determined by Algorithm.

- o Replay Window. The replay protection window for messages received, equivalent to the functionality described in Section 4.1.2.6 of [RFC6347].

The 3-tuple (Cid, Sender ID, Sender Sequence Number) is called Transaction Identifier (Tid), and SHALL be unique for each COSE object and server. The Tid is used as a unique challenge in the COSE object of the protected CoAP request. The Tid is part of the Additional Authenticated Data (AAD, see Section 5) of the protected CoAP response message, which is how the challenge becomes signed by the server.

The client and server may change roles while maintaining the same security context. The former server will then make the request using the Sender Context, the former client will verify the request using its Recipient Context etc.

3.2. Security Context Establishment

This section aims at describing how to establish the security context, given some input parameters. The input parameters, which are established in a previous phase, are:

- Context Identifier (Cid)
- Algorithm (Alg)
- Base Key (base_key)
- Sender ID
- Recipient ID
- Replay Window (optionally)

These are included unchanged in the security context. We give below some indications on how applications should select these parameters. Moreover, the following parameters are established as described below:

- Sender Key
- Sender IV
- Sender Sequence Number
- Recipient Key
- Recipient IV
- Recipient Sequence Number
- Replay Window

3.2.1. Derivation of Sender Key/IV, Recipient Key/IV

Given a common shared secret material and a common key derivation function, the client and server can derive the security context necessary to run OSCOAP. The derivation procedure described here MUST NOT be executed more than once on a set of common secret material. Also, the same base_key SHOULD NOT be used in different security contexts (identified by different Cids).

The procedure assumes that the common shared secret material is uniformly random and that the key derivation function is HKDF

[RFC5869]. This is for example the case after having used EDHOC [I-D.selander-ace-cose-ecdhe].

Assumptions:

- o The hash function, denoted HKDF, is the HMAC based key derivation function defined in [RFC5869] with specified hash function

- o The common shared secret material, denoted base_key, is uniformly pseudo-random of length at least equal to the output of the specified hash function

The security context parameters Sender Key/IV, Recipient Key/IV SHALL be derived using the HKDF-Expand primitive [RFC5869]:

output parameter = HKDF-Expand(base_key, info, key_length),

where:

- o base_key is defined above

- o info = Cid || Sender ID/Recipient ID || "IV"/"Key" || Algorithm || key_length

- o key_length is the key size of the AEAD algorithm

The Sender/Recipient Key shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "Key", the Algorithm and the key_length. The Sender/Recipient IV shall be derived using the Cid concatenated with the Sender/Recipient ID, the label "IV", the Algorithm and the key_length.

For example, for the algorithm AES-CCM-64-64-128 (see Section 10.2 in [I-D.ietf-cose-msg]), key_length for the keys is 128 bits and key_length for the context IVs is 56 bits.

3.2.2. Sequence Numbers and Replay Window

The values of the Sequence Numbers are initialized to 0 during establishment of the security context. The default Replay Window size of 64 is used if no input parameter is provided in the set up phase.

3.2.3. Context Identifier and Sender/Recipient ID

As mentioned, Cid, Sender ID and Recipient ID are established in a previous phase. How this is done is application specific, but some guidelines are given in this section.

It is RECOMMENDED that the application uses 64-bits long pseudo-random Cids, in order to have globally unique Context Identifiers. Cid SHOULD be unique in the sets of all security contexts used by all the endpoints. If it is not the case, it is the role of the application to specify how to handle collisions.

In the same phase during which the Cid is established in the endpoint, the application informs the endpoint what resource can be accessed using the corresponding security context. The granularity of that is decided by the application (resource, host, etc). The endpoint SHALL save the association resource-Cid, in order to be able to retrieve the correct security context to access a resource.

The Sender ID and Recipient ID are also established in the endpoint during the previous set up phase. The application SHOULD make sure that these identifiers are locally unique in the set of all endpoints using the same security context. If it is not the case, it is again the role of the application to specify how to handle collisions.

In case of EDHOC [I-D.selander-ace-cose-ecdhe]) the Cid is the hash of the messages exchanged.

4. Protected CoAP Message Fields

This section defines how the CoAP message fields are protected. OSCOAP protects as much of the unprotected CoAP message as possible, while still allowing forward proxy operations [I-D.hartke-core-e2e-security-reqs].

The CoAP Payload SHALL be encrypted and integrity protected.

The CoAP Header fields Version and Code SHALL be integrity protected but not encrypted. The CoAP Message Layer parameters, Type and Message ID, as well as Token and Token Length SHALL neither be integrity protected nor encrypted.

Protection of CoAP Options can be summarized as follows:

- To prevent information leakage, Uri-Path and Uri-Query SHALL be encrypted. As a consequence, if Proxy-Uri is used, those parts of the URI SHALL be removed from the Proxy-Uri. The CoAP Options Uri-Host, Uri-Port, Proxy-Uri, and Proxy-Scheme SHALL neither be encrypted, nor integrity protected (cf. protection of the effective request URI in Section 5.2).

- The other CoAP options SHALL be encrypted and integrity protected.

A summary of which options are encrypted or integrity protected is shown in Figure 4.

No.	C	U	N	R	Name	Format	Length	E	D
1	x			x	If-Match	opaque	0-8	x	
3	x	x	-		Uri-Host	string	1-255		
4				x	ETag	opaque	1-8	x	
5	x				If-None-Match	empty	0	x	
6		x	-		Observe	uint	0-3	x	x
7	x	x	-		Uri-Port	uint	0-2		
8				x	Location-Path	string	0-255	x	
11	x	x	-	x	Uri-Path	string	0-255	x	
12					Content-Format	uint	0-2	x	
14		x	-		Max-Age	uint	0-4	x	x
15	x	x	-	x	Uri-Query	string	0-255	x	
17	x				Accept	uint	0-2	x	
20				x	Location-Query	string	0-255	x	
23	x	x	-	-	Block2	uint	0-3	x	x
27	x	x	-	-	Block1	uint	0-3	x	x
28			x		Size2	unit	0-4	x	x
35	x	x	-		Proxy-Uri	string	1-1034		
39	x	x	-		Proxy-Scheme	string	1-255		
60			x		Size1	uint	0-4	x	x

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable,
E=Encrypt and Integrity Protect, D=Duplicate.

Figure 4: Protection of CoAP Options

Unless specified otherwise, CoAP options not listed in Figure 4 SHALL be encrypted and integrity protected.

The encrypted options are in general omitted from the protected CoAP message and not visible to intermediary nodes (see Section 6.2 and Section 6.4). Hence the actions resulting from the use of corresponding options is analogous to the case of communicating directly with the endpoint. For example, a client using an ETag option will not be served by a proxy.

However, some options which are encrypted need to be readable in the protected CoAP message to support certain proxy functions. A CoAP option which may be both encrypted in the COSE object of the protected CoAP message, and also unencrypted as CoAP option in the protected CoAP message, is called "duplicate". The "encrypted" value of a duplicate option is intended for the destination endpoint and

the "unencrypted" value is intended for a proxy. The unencrypted value is not integrity protected.

- o The Max-Age option is duplicate. The unencrypted Max-Age SHOULD have value zero to prevent caching of responses. The encrypted Max-Age is used as defined in [RFC7252] taking into account that it is not accessible to proxies.

- o The Observe option is duplicate. If Observe is used, then the encrypted Observe and the unencrypted Observe SHALL have the same value. The Observe option as used here targets the requirements on forwarding of [I-D.hartke-core-e2e-security-reqs] (Section 2.2.1.2).

- o The block options Block1 and Block2 are duplicate. The encrypted block options is used for end-to-end secure fragmentation of payload into blocks and protected information about the fragmentation (block number, last block, etc.). The MAC from each block is included in the calculation of the MAC for the next block’s (see Section 5.2). In this way, each block in ordered sequence from the first block can be verified as it arrives. The unencrypted block option allows for arbitrary proxy fragmentation operations which cannot be verified by the endpoints. An intermediary node can generate an arbitrarily long sequence of blocks. However, since it is possible to protect fragmentation of large messages, there SHALL be a security policy defining a maximum unfragmented message size such that messages exceeding this size SHALL be fragmented by the sending endpoint. Hence an endpoint receiving fragments of a message that exceeds maximum message size SHALL discard this message.

- o The size options Size1 and Size2 are duplicate, analogously to the block options.

Specifications of new CoAP options SHOULD specify how they are processed with OSCOAP. New COAP options SHALL be encrypted and integrity protected. New COAP options SHOULD NOT be duplicate unless a forwarding proxy needs to read the option. If an option is registered as duplicate, the duplicate value SHOULD NOT be the same as the end-to-end value, unless the proxy is required by specification to be able to read the end-to-end value.

5. The COSE Object

This section defines how to use the COSE format [I-D.ietf-cose-msg] to wrap and protect data in the unprotected CoAP message. OSCOAP uses the COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm.

The mandatory to support AEAD algorithm is AES-CCM-64-64-128 defined in Section 10.2 of [I-D.ietf-cose-msg]. For AES-CCM-64-64-128 the length of Sender Key and Recipient Key SHALL be 128 bits, the length of IV, Sender IV, and Recipient IV SHALL be 7 bytes, and the maximum Sender Sequence Number and Recipient Sequence Number SHALL be 2^56-1. The IV is constructed using a Partial IV exactly like in Section 3.1 of [I-D.ietf-cose-msg], i.e. by padding the Sender Sequence Number or the Recipient Sequence Number with zeroes and XORing it with the Sender IV or Recipient IV, respectively.

Since OSCOAP only makes use of a single COSE structure, there is no need to explicitly specify the structure, and OSCOAP uses the untagged version of the COSE_Encrypt0 structure (Section 2. of [I-D.ietf-cose-msg]). If the COSE object has a different structure, the recipient MUST reject the message, treating it as malformed.

We denote by Plaintext the data that is encrypted and integrity protected, and by Additional Authenticated Data (AAD) the data that is integrity protected only, in the COSE object.

The fields of COSE_Encrypt0 structure are defined as follows (see example in Appendix C.4).

- o The "Headers" field is formed by:
  - * The "protected" field, which SHALL include:
    - + The "Partial IV" parameter. The value is set to the Sender Sequence Number. The Partial IV is a byte string (type: bstr), where the length is the minimum length needed to encode the sequence number. An Endpoint that receives a COSE object with a sequence number encoded with leading zeroes (i.e. longer than the minimum needed length) SHALL reject the corresponding message as malformed.
    - + If the message is a CoAP request, the "kid" parameter. The value is set to the Context Identifier (see Section 3).
    - + Optionally, the parameter called "sid", defined below. The value is set to the Sender ID (see Section 3). Note that since this parameter is sent in clear, privacy issues SHOULD be considered by the application defining the Sender ID.
  - * The "unprotected" field, which SHALL be empty.
- o The "cipher text" field is computed from the Plaintext (see Section 5.1) and the Additional Authenticated Data (AAD) (see

Section 5.2) and encoded as a byte string (type: bstr), following Section 5.2 of [I-D.ietf-cose-msg].

sid: This parameter is used to identify the sender of the message. Applications MUST NOT assume that 'sid' values are unique. This is not a security critical field. For this reason, it can be placed in the unprotected headers bucket.

name	label	value type	value registry	description
sid	TBD	bstr		Sender identifier

Table 1: Additional COSE Header Parameter

5.1. Plaintext

The Plaintext is formatted as a CoAP message without Header (see Figure 5) consisting of:

- o all CoAP Options present in the unprotected message which are encrypted (see Section 4), in the order as given by the Option number (each Option with Option Header including delta to previous included encrypted option); and

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Options to Encrypt (if any) ...                            ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...                       ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 (only if there
   is payload)
```

Figure 5: Plaintext

5.2. Additional Authenticated Data

The Additional Authenticated Data ("Enc_structure") as described is Section 5.3 of [I-D.ietf-cose-msg] includes:

- o the "context" parameter, which has value "Encrypted"

- o the "protected" parameter, which includes the "protected" part of the "Headers" field;

- o the "external_aad" is a serialized CBOR array (see Figure 8) that contains, in the given order:

  * ver: uint, contains the CoAP version number of the unprotected CoAP message, as defined in Section 3 of [RFC7252]

  * code: bstr, contains is the CoAP Code of the unprotected CoAP message, as defined in Section 3 of [RFC7252].

  * alg: bstr, contains the serialized Algorithm from the security context used for the exchange (see Section 3.1);

  * request-uri: tstr, contains the plaintext "effective" request URI composed from the request scheme and Uri-* options according to the method described in Section 6.5 of [RFC7252], if the message is a CoAP request;

  * transaction-id: bstr, only included if the message to protect or verify is a CoAP response, contains the Transaction Identifier (Tid) of the associated CoAP request (see Section 3). Note that the Tid is the 3-tuple (Cid, Sender ID, Sender Sequence Number) for the endpoint sending the request and verifying the response; which means that for the endpoint sending the response, the Tid has value (Cid, Recipient ID, seq), where seq is the value of the "Partial IV" in the COSE object of the request (see Section 5); and

  * mac-previous-block: bstr, contains the MAC of the message containing the previous block in the sequence, as enumerated by Block1 in the case of a request and Block2 in the case of a response, if the message is fragmented using a block option [RFC7959].

```
external_aad_req = [
    ver : uint,
    code : bstr,
    alg : bstr,
    request-uri : tstr,
    ? mac-previous-block : bstr
]
```

Figure 6: external_aad for a request

```
external_aad_resp = [
    ver : uint,
    code : bstr,
    alg : bstr,
    transaction-id : bstr,
    ? mac-previous-block : bstr
]
```

Figure 7: external_aad for a response

```
external_aad = external_aad_req / external_aad_resp
```

Figure 8: external_aad

The encryption process is described in Section 5.3 of [I-D.ietf-cose-msg].

6. Protecting CoAP Messages

6.1. Replay and Freshness Protection

In order to protect from replay of messages and verify freshness, a CoAP endpoint SHALL maintain a Sender Sequence Number, and a Recipient Sequence Number associated to a security context, which is identified with a Context Identifier (Cid). The two sequence numbers are the highest sequence number the endpoint has sent and the highest sequence number the endpoint has received. An endpoint uses the Sender Sequence Number to protect messages to send and the Recipient Sequence Number to verify received messages, as described in Section 3.

Depending on use case and ordering of messages provided by underlying layers, an endpoint MAY maintain a sliding replay window for Sequence Numbers of received messages associated to each Cid. In case of reliable transport, the receiving endpoint MAY require that the Sequence Number of a received message equals last Sequence Number + 1.

A receiving endpoint SHALL verify that the Sequence Number received in the COSE object has not been received before in the security context identified by the Cid. The receiving endpoint SHALL also reject messages with a sequence number greater than 2^56-1.

OSCOAP is a challenge-response protocol, where the response is verified to match a prior request, by including the unique transaction identifier (Tid as defined in Section 3) of the request in the Additional Authenticated Data of the response message.

If a CoAP server receives a request with the Object-Security option, then the server SHALL include the Tid of the request in the AAD of the response, as described in Section 6.4.

If the CoAP client receives a response with the Object-Security option, then the client SHALL verify the integrity of the response, using the Tid of its own associated request in the AAD, as described in Section 6.5.

6.2. Protecting the Request

Given an unprotected CoAP request, including header, options and payload, the client SHALL perform the following steps to create a protected CoAP request using a security context associated with the target resource (see Section 3.2.3).

1. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the client MUST NOT process any requests with the given security context. The client SHOULD acquire a new security context (and consequently inform the server about it) before this happens. The latter is out of scope of this memo.

2. Compute the COSE object as specified in Section 5

   * the IV in the AEAD is created by XORing the Sender IV (context IV) with the Sender Sequence Number (partial IV).

   * If the block option is used, the AAD includes the MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-sent fragment to compute the following.

   * Note that the ’sid’ field containing the Sender ID is included in the COSE object (Section 5) if the application needs it.

3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload, based on the unprotected CoAP message:

   * The CoAP header is the same as the unprotected CoAP message.

   * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.

* If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

4. Store in memory the association Token - Cid. The Client SHALL be able to find the correct security context used to protect the request and verify the response with use of the Token of the message exchange.

6.3. Verifying the Request

A CoAP server receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Context Identifier in the "kid" parameter in the received COSE object:

1. Verify the Sequence Number in the Partial IV parameter, as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the server MUST stop processing the request.

2. Recreate the Additional Authenticated Data, as described in Section 5.

   * If the block option is used, the AAD includes the MAC from the previous fragment received (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-received fragment to compute the following.

3. Compose the IV by XORing the Recipient IV (context IV) with the Partial IV parameter, received in the COSE Object.

4. Retrieve the Recipient Key.

5. Verify and decrypt the message. If the verification fails, the server MUST stop processing the request.

6. If the message verifies, update the Recipient Sequence Number or Replay Window, as described in Section 6.1.

7. Restore the unprotected request by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

6.4. Protecting the Response

A server receiving a valid request with a protected CoAP message (i.e. containing an Object-Security option) SHALL respond with a protected CoAP message.

Given an unprotected CoAP response, including header, options, and payload, the server SHALL perform the following steps to create a protected CoAP response, using the security context identified by the Context Identifier of the received request:

1. Increment the Sender Sequence Number by one (note that this means that sequence number 0 is never used). If the Sender Sequence Number exceeds the maximum number for the AEAD algorithm, the server MUST NOT process any more responses with the given security context. The server SHOULD acquire a new security context (and consequently inform the client about it) before this happens. The latter is out of scope of this memo.

2. Compute the COSE object as specified in Section Section 5

   * The IV in the AEAD is created by XORing the Sender IV (context IV) and the Sender Sequence Number.

   * If the block option is used, the AAD includes the MAC from the previous fragment sent (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-sent fragment to compute the following.

3. Format the protected CoAP message as an ordinary CoAP message, with the following Header, Options, and Payload based on the unprotected CoAP message:

   * The CoAP header is the same as the unprotected CoAP message.

   * The CoAP options which are encrypted and not duplicate (Section 4) are removed. Any duplicate option which is present has its unencrypted value. The Object-Security option is added.

   * If the message type of the unprotected CoAP message does not allow Payload, then the value of the Object-Security option is the COSE object. If the message type of the unprotected CoAP message allows Payload, then the Object-Security option is empty and the Payload of the protected CoAP message is the COSE object.

Note the differences between generating a protected request, and a protected response, for example whether "kid" is present in the header, or whether Destination URI or Tid is present in the AAD, of the COSE object.

6.5. Verifying the Response

A CoAP client receiving a message containing the Object-Security option SHALL perform the following steps, using the security context identified by the Token of the received response:

1. Verify the Sequence Number in the Partial IV parameter as described in Section 6.1. If it cannot be verified that the Sequence Number has not been received before, the client MUST stop processing the response.

2. Recreate the Additional Authenticated Data as described in Section 5.

   * If the block option is used, the AAD includes the MAC from the previous fragment received (from the second fragment and following) Section 5.2. This means that the endpoint MUST store the MAC of each last-received fragment to compute the following.

3. Compose the IV by XORing the Recipient IV (context IV) with the Partial IV parameter, received in the COSE Object.

4. Retrieve the Recipient Key.

5. Verify and decrypt the message. If the verification fails, the client MUST stop processing the response.

6. If the message verifies, update the Recipient Sequence Number or Replay Window, as described in Section 6.1.

7. Restore the unprotected response by adding any decrypted options or payload from the plaintext. Any duplicate options (Section 4) are overwritten. The Object-Security option is removed.

7. Security Considerations

In scenarios with intermediary nodes such as proxies or brokers, transport layer security such as DTLS only protects data hop-by-hop. As a consequence the intermediary nodes can read and modify information. The trust model where all intermediate 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 brittle.

DTLS protects hop-by-hop the entire CoAP message, including header, options, and payload. OSCOAP protects end-to-end the payload, and all information in the options and header, that is not required for forwarding (see Section 4). DTLS and OSCOAP can be combined, thereby enabling end-to-end security of CoAP payload, in combination with hop-by-hop protection of the entire CoAP message, during transport between end-point and intermediary node.

The CoAP message layer, however, cannot be protected end-to-end through intermediary devices since the parameters Type and Message ID, as well as Token and Token Length may be changed by a proxy. Moreover, messages that are not possible to verify should for security reasons not always be acknowledged but in some cases be silently dropped. This would not comply with CoAP message layer, but does not have an impact on the application layer security solution, since message layer is excluded from that.

The use of COSE to protect CoAP messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common shared secret material and key derivation function in client and server. EDHOC [I-D.selander-ace-cose-ecdhe] describes an augmented Diffie-Hellman key exchange to produce forward secret keying material and agree on crypto algorithms necessary for OSCOAP, authenticated with pre-established credentials. These pre-established credentials may, in turn, be provisioned using a trusted third party such as described in the OAuth-based ACE framework [I-D.ietf-ace-oauth-authz]. An OSCOAP profile of ACE is described in [I-D.seitz-ace-oscoap-profile].

For symmetric encryption it is required to have a unique IV for each message, for which the sequence numbers in the COSE message field "Partial IV" is used. The context IVs (Sender IV and Recipient IV) SHOULD be established between sender and recipient before the message is sent, for example using the method in [I-D.selander-ace-cose-ecdhe], to avoid the overhead of sending it in each message.

The MTI AEAD algorithm AES-CCM-64-64-128 is selected for broad applicability in terms of message size (2^64 blocks) and maximum no. messages (2^56-1). For 128 bit CCM*, use instead AES-CCM-16-64-128 [I-D.ietf-cose-msg].

If the recipient accepts any sequence number larger than the one previously received (less than the maximum sequence number), then the problem of sequence number synchronization is avoided. With reliable transport it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. The alternatives to sequence numbers have their issues: very constrained devices may not be able to support accurate time, or to generate and store large numbers of random IVs. 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.

The encrypted block options enable the sender to split large messages into protected fragments such that the receiving node can verify blocks before having received the complete message. In order to protect from attacks replacing fragments from a different message with the same block number between same endpoints and same resource at roughly the same time, the MAC from the message containing one block is included in the external_aad of the message containing the next block.

The unencrypted block options allow for arbitrary proxy fragmentation operations which cannot be verified by the endpoints, but can by policy be restricted in size since the encrypted options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

8. Privacy Considerations

Privacy threats executed through intermediate nodes are considerably reduced by means of OSCOAP. End-to-end integrity protection and encryption of CoAP payload and all options that are not used for forwarding, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

CoAP headers sent in plaintext allow for example matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis.

9. IANA Considerations

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

9.1. Sid Registration

IANA is requested to enter a new parameter entitled "sid" to the registry "COSE Header Parameters". The parameter is defined in Table 1.

9.2. CoAP Option Number Registration

The Object-Security option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD	Object-Security	[[this document]]

9.3. Media Type Registrations

The "application/oscon" media type is added to the Media Types registry:

Type name: application

Subtype name: cose

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[this document]].

Interoperability considerations: N/A

Published specification: [[this document]]

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

9.4. CoAP Content Format Registration

The "application/oscon" content format is added to the CoAP Content Format registry:

Media type	Encoding	ID	Reference
application/oscon	-	70	[[this document]]

10. Acknowledgments

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. We are grateful to Malisa Vucinic and Marco Tiloca for providing helpful and timely reviews of previous versions of the draft. We are also grateful to Carsten Bormann and Jim Schaad for providing input and interesting discussions.

11. References

11.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-20 (work in progress), October 2016.

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

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

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

11.2. Informative References

[I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-01 (work in progress), July 2016.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-02 (work in progress), June 2016.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-04 (work in progress), August 2016.

[I-D.seitz-ace-oscoap-profile]
Seitz, L., "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-00 (work in progress), July 2016.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-02 (work in progress), July 2016.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

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

Appendix A. Overhead

OSCOAP transforms an unprotected CoAP message to a protected CoAP message, and the protected CoAP message is larger than the unprotected CoAP message. This appendix illustrates the message expansion.

A.1. Length of the Object-Security Option

The protected CoAP message contains the COSE object. The COSE object is included in the payload if the message type of the unprotected CoAP message allows payload or else in the Object-Security option. In the former case the Object-Security option is empty. So the length of the Object-Security option is either zero or the size of the COSE object, depending on whether the CoAP message allows payload or not.

Length of Object-Security option = { 0, size of COSE Object }

A.2. Size of the COSE Object

The size of the COSE object is the sum of the sizes of

- o the Header parameters,
- o the Cipher Text (excluding the Tag),
- o the Tag, and
- o data incurred by the COSE format itself (including CBOR encoding).

Let’s analyse the contributions one at a time:

- o The header parameters of the COSE object are the Context Identifier (Cid) and the Sequence Number (Seq) (also known as the Transaction Identifier (Tid)) if the message is a request, and Seq only if the message is a response (see Section 5).
  - * The size of Cid depends on the number of simultaneous clients, as discussed in Section 3.2
  - * The size of Seq is variable, and increases with the number of messages exchanged.
  - * As the IV is generated from the padded Sequence Number and a previously agreed upon context IV it is not required to send the whole IV in the message.

- o The Cipher Text, excluding the Tag, is the encryption of the payload and the encrypted options Section 4, which are present in the unprotected CoAP message.

- o The size of the Tag depends on the Algorithm. For example, for the algorithm AES-CCM-64-64-128, the Tag is 8 bytes.

- o The overhead from the COSE format itself depends on the sizes of the previous fields, and is of the order of 10 bytes.

A.3. Message Expansion

The message expansion is not the size of the COSE object. The cipher text in the COSE object is encrypted payload and options of the unprotected CoAP message - the plaintext of which is removed from the protected CoAP message. Since the size of the cipher text is the same as the corresponding plaintext, there is no message expansion due to encryption; payload and options are just represented in a different way in the protected CoAP message:

- o The encrypted payload is in the payload of the protected CoAP message

- o The encrypted options are in the Object-Security option or within the payload.

Therefore the OSCOAP message expansion is due to Cid (if present), Seq, Tag, and COSE overhead:

```
Message Overhead = Cid + Seq + Tag + COSE Overhead
```

Figure 9: OSCOAP message expansion

A.4. Example

This section gives an example of message expansion in a request with OSCOAP.

In this example we assume an extreme 4-byte Cid, based on the assumption of an ACE deployment with billions of clients requesting access to this particular server. (A typical Cid, will be 1-2 byte as is discussed in Appendix A.2.)

- o Cid: 0xa1534e3c

In the example the sequence number is 225, requiring 1 byte to encode. (The size of Seq could be larger depending on how many messages that has been sent as is discussed in Appendix A.2.)

o Seq: 225

The example is based on AES-CCM-64-64-128.

o Tag is 8 bytes

The COSE object is represented in Figure 10 using CBOR's diagnostic notation.

```
[
  h'a20444a1534e3c0641e2', # protected:
                             {04:h'a1534e3c',
                              06:h'e2'}
  {},                      # unprotected: -
  Tag                      # cipher text + 8 byte authentication tag
]
```

Figure 10: Example of message expansion

Note that the encrypted CoAP options and payload are omitted since we target the message expansion (see Appendix A.3). Therefore the size of the COSE Cipher Text equals the size of the Tag, which is 8 bytes.

The COSE object encodes to a total size of 22 bytes, which is the message expansion in this example. The COSE overhead in this example is 22 - (4 + 1 + 8) = 9 bytes, according to the formula in Figure 9. Note that in this example two bytes in the COSE overhead are used to encode the length of Cid and the length of Seq.

Figure 11 summarizes these results.

Tid	Tag	COSE OH	Message OH
5 bytes	8 bytes	9 bytes	22 bytes

Figure 11: Message overhead for a 5-byte Tid and 8-byte Tag.

Appendix B. Examples

This section gives examples of OSCOAP. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the COSE message format.

B.1. Secure Access to Sensor

Here is an example targeting the scenario in the Section 2.2.1. - Forwarding of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting the alarm status from a server. In the request, CoAP option Uri-Path is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected (see Section 4). In the response, the CoAP Payload is encrypted and integrity protected, and the CoAP header fields Code and Version are integrity protected.

```
   Client  Proxy  Server
      |      |      |
      +----->|      |            Code: 0.01 (GET)
      | GET  |      |           Token: 0x8c
      |      |      | Object-Security: [cid:5fdc, seq:42,
      |      |      |                   {Uri-Path:"alarm_status"},
      |      |      |                   <Tag>]
      |      |      |         Payload: -
      |      |      |
      |      +----->|            Code: 0.01 (GET)
      |      | GET  |           Token: 0x7b
      |      |      | Object-Security: [cid:5fdc, seq:42,
      |      |      |                   {Uri-Path:"alarm_status"},
      |      |      |                   <Tag>]
      |      |      |         Payload: -
      |      |      |
      |      |<-----+            Code: 2.05 (Content)
      |      | 2.05 |           Token: 0x7b
      |      |      |         Max-Age: 0
      |      |      | Object-Security: -
      |      |      |         Payload: [seq:56, {"OFF"}, <Tag>]
      |      |      |
      |<-----+      |            Code: 2.05 (Content)
      | 2.05 |      |           Token: 0x8c
      |      |      |         Max-Age: 0
      |      |      | Object-Security: -
      |      |      |         Payload: [seq:56, {"OFF"}, <Tag>]
      |      |      |
```

Figure 12: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicate a COSE object. The brackets { ... } indicate encrypted data.

Since the unprotected request message (GET) has no payload, the Object-Security option carries the COSE object as its value. Since the unprotected response message (Content) has payload ("OFF"), the COSE object (indicated with [...]) is carried as the CoAP payload.

The COSE header of the request contains a Context Identifier (cid:5fdc), indicating which security context was used to protect the message and a Sequence Number (seq:42).

The option Uri-Path (alarm_status) and payload ("OFF") are formatted as indicated in Section 5, and encrypted in the COSE Cipher Text (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence Number has not been received before and that the response message is generated as a response to the sent request message (see Section 6.1).

B.2. Secure Subscribe to Sensor

Here is an example targeting the scenario in the Forwarding with observe case of [I-D.hartke-core-e2e-security-reqs]. The example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), and first receiving the value 220 mg/dl, and then a second reading with value 180 mg/dl. The CoAP options Observe, Uri-Path, Content-Format, and Payload are encrypted and integrity protected, and the CoAP header field Code is integrity protected (see Section 4).

```
Client  Proxy  Server
   |      |      |
   +----->|      |            Code: 0.01 (GET)
   | GET  |      |           Token: 0x83
   |      |      |         Observe: 0
   |      |      | Object-Security: [cid:ca, seq:15b7, {Observe:0,
   |      |      |                   Uri-Path:"glucose"}, <Tag>]
   |      |      |         Payload: -
   |      |      |
   |      +----->|            Code: 0.01 (GET)
   |      | GET  |           Token: 0xbe
   |      |      |         Observe: 0
   |      |      | Object-Security: [cid:ca, seq:15b7, {Observe:0,
   |      |      |                   Uri-Path:"glucose"}, <Tag>]
   |      |      |         Payload: -
   |      |      |
   |      |<-----+            Code: 2.05 (Content)
   |      | 2.05 |           Token: 0xbe
   |      |      |         Max-Age: 0
   |      |      |         Observe: 1
   |      |      | Object-Security: -
   |      |      |         Payload: [seq:32c2, {Observe:1,
   |      |      |                   Content-Format:0, "220"}, <Tag>]
```

```
      |      |      |
      |<-----+      |            Code: 2.05 (Content)
      | 2.05 |      |           Token: 0x83
      |      |      |         Max-Age: 0
      |      |      |         Observe: 1
      |      |      | Object-Security: -
      |      |      |         Payload: [seq:32c2, {Observe:1,
      |      |      |                   Content-Format:0, "220"}, <Tag>]
    ...    ...    ...
      |      |      |
      |      |<-----+            Code: 2.05 (Content)
      |      | 2.05 |           Token: 0xbe
      |      |      |         Max-Age: 0
      |      |      |         Observe: 2
      |      |      | Object-Security: -
      |      |      |         Payload: [seq:32c6, {Observe:2,
      |      |      |                   Content-Format:0, "180"}, <Tag>]
      |      |      |
      |<-----+      |            Code: 2.05 (Content)
      | 2.05 |      |           Token: 0x83
      |      |      |         Max-Age: 0
      |      |      |         Observe: 2
      |      |      | Object-Security: -
      |      |      |         Payload: [seq:32c6, {Observe:2,
      |      |      |                   Content-Format:0, "180"}, <Tag>]
      |      |      |
```

Figure 13: Indication of CoAP GET protected with OSCOAP. The brackets [...] indicates COSE object. The bracket { ... } indicates encrypted data.

Since the unprotected request message (GET) allows no payload, the COSE object (indicated with [...]) is carried in the Object-Security option value. Since the unprotected response message (Content) has payload, the Object-Security option is empty, and the COSE object is carried as the payload.

The COSE header of the request contains a Context Identifier (cid:ca), indicating which security context was used to protect the message and a Sequence Number (seq:15b7).

The options Observe, Content-Format and the payload are formatted as indicated in Section 5, and encrypted in the COSE cipher text (indicated with { ... }).

The server verifies that the Sequence Number has not been received before (see Section 6.1). The client verifies that the Sequence

Number has not been received before and that the response message is generated as a response to the subscribe request.

Appendix C. Object Security of Content (OSCON)

OSCOAP protects message exchanges end-to-end between a certain client and a certain server, targeting the security requirements for forward proxy of [I-D.hartke-core-e2e-security-reqs]. In contrast, many use cases require one and the same message to be protected for, and verified by, multiple endpoints, see caching proxy section of [I-D.hartke-core-e2e-security-reqs]. Those security requirements can be addressed by protecting essentially the payload/content of individual messages using the COSE format ([I-D.ietf-cose-msg]), rather than the entire request/response message exchange. This is referred to as Object Security of Content (OSCON).

OSCON transforms an unprotected CoAP message into a protected CoAP message in the following way: the payload of the unprotected CoAP message is wrapped by a COSE object, which replaces the payload of the unprotected CoAP message. We call the result the "protected" CoAP message.

The unprotected payload shall be the plaintext/payload of the COSE object. The ’protected’ field of the COSE object ’Headers’ shall include the context identifier, both for requests and responses. If the unprotected CoAP message includes a Content-Format option, then the COSE object shall include a protected ’content type’ field, whose value is set to the unprotected message Content-Format value. The Content-Format option of the protected CoAP message shall be replaced with "application/oscon" (Section 9)

The COSE object shall be protected (encrypted) and verified (decrypted) as described in ([I-D.ietf-cose-msg]).

In the case of symmetric encryption, the same key and IV shall not be used twice. Sequence numbers for partial IV as specified for OSCOAP may be used for replay protection as described in Section 6.1. The use of time stamps in the COSE header parameter ’operation time’ [I-D.ietf-cose-msg] for freshness may be used.

OSCON shall not be used in cases where CoAP header fields (such as Code or Version) or CoAP options need to be integrity protected or encrypted. OSCON shall not be used in cases which require a secure binding between request and response.

The scenarios in Sections 3.3 - 3.5 of [I-D.hartke-core-e2e-security-reqs] assume multiple recipients for a particular content. In this case the use of symmetric keys does not

provide data origin authentication. Therefore the COSE object should in general be protected with a digital signature.

C.1. Overhead OSCON

In general there are four different kinds of ciphersuites that need to be supported: message authentication code, digital signature, authenticated encryption, and symmetric encryption + digital signature. The use of digital signature is necessary for applications with many legitimate recipients of a given message, and where data origin authentication is required.

To distinguish between these different cases, the tagged structures of COSE are used (see Section 2 of [I-D.ietf-cose-msg]).

The size of the COSE message for selected algorithms are detailed in this section.

The size of the header is shown separately from the size of the MAC/ signature. A 4-byte Context Identifier and a 1-byte Sequence Number are used throughout all examples, with these values:

- Cid: 0xa1534e3c

- Seq: 0xa3

For each scheme, we indicate the fixed length of these two parameters ("Cid+Seq" column) and of the Tag ("MAC"/"SIG"/"TAG"). The "Message OH" column shows the total expansions of the CoAP message size, while the "COSE OH" column is calculated from the previous columns following the formula in Figure 9.

Overhead incurring from CBOR encoding is also included in the COSE overhead count.

To make it easier to read, COSE objects are represented using CBOR’s diagnostic notation rather than a binary dump.

C.2. MAC Only

This example is based on HMAC-SHA256, with truncation to 8 bytes (HMAC 256/64).

Since the key is implicitly known by the recipient, the COSE_Mac0_Tagged structure is used (Section 6.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
996(                          # COSE_Mac0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                       # unprotected
    h'',                      # payload
    MAC                       # truncated 8-byte MAC
  ]
)
```

This COSE object encodes to a total size of 26 bytes.

Figure 14 summarizes these results.

Structure	Tid	MAC	COSE OH	Message OH
COSE_Mac0_Tagged	5 B	8 B	13 B	26 B

Figure 14: Message overhead for a 5-byte Tid using HMAC 256/64

C.3. Signature Only

This example is based on ECDSA, with a signature of 64 bytes.

Since only one signature is used, the COSE_Sign1_Tagged structure is used (Section 4.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
997(                          # COSE_Sign1_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                       # unprotected
    h'',                      # payload
    SIG                       # 64-byte signature
  ]
)
```

This COSE object encodes to a total size of 83 bytes.

Figure 15 summarizes these results.

Structure	Tid	SIG	COSE OH	Message OH
COSE_Sign1_Tagged	5 B	64 B	14 B	83 bytes

Figure 15: Message overhead for a 5-byte Tid using 64 byte ECDSA signature.

C.4. Authenticated Encryption with Additional Data (AEAD)

This example is based on AES-CCM with the MAC truncated to 8 bytes.

It is assumed that the IV is generated from the Sequence Number and some previously agreed upon context IV. This means it is not required to explicitly send the whole IV in the message.

Since the key is implicitly known by the recipient, the COSE_Encrypt0_Tagged structure is used (Section 5.2 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
993(                        # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {},                     # unprotected
    TAG                     # cipher text + truncated 8-byte TAG
  ]
)
```

This COSE object encodes to a total size of 25 bytes.

Figure 16 summarizes these results.

Structure	Tid	TAG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	12 B	25 bytes

Figure 16: Message overhead for a 5-byte Tid using AES_128_CCM_8.

C.5. Symmetric Encryption with Asymmetric Signature (SEAS)

This example is based on AES-CCM and ECDSA with 64 bytes signature. The same assumption on the security context as in Appendix C.4. COSE defines the field 'counter signature w/o headers' that is used here to sign a COSE_Encrypt0_Tagged message (see Section 3 of [I-D.ietf-cose-msg]).

The object in COSE encoding gives:

```
993(                          # COSE_Encrypt0_Tagged
  [
    h'a20444a1534e3c0641a3', # protected:
                               {04:h'a1534e3c',
                                06:h'a3'}
    {9:SIG},                  # unprotected:
                                09: 64 bytes signature
    TAG                       # cipher text + truncated 8-byte TAG
  ]
)
```

This COSE object encodes to a total size of 92 bytes.

Figure 17 summarizes these results.

Structure	Tid	TAG	SIG	COSE OH	Message OH
COSE_Encrypt0_Tagged	5 B	8 B	64 B	15 B	92 B

Figure 17: Message overhead for a 5-byte Tid using AES-CCM countersigned with ECDSA.

Authors' Addresses

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Ludwig Seitz
SICS Swedish ICT
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig@sics.se