Using The Delegated CoAP Authentication and Authorization Framework (DCAF) With CBOR Encoded Message Syntax
draft-bergmann-ace-dcaf-cose-00

Abstract

This specification defines a profile for the Delegated CoAP Authentication and Authorization Framework (DCAF) that facilitates client authentication and authorization in a constrained environment using the CBOR Encoded Message Syntax.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

The Delegated CoAP Authentication and Authorization Framework (DCAF) is designed to be agnostic of the actual mechanism being used to secure the communication between the ACE actors. While the original specification [I-D.gerdes-ace-dcaf-authorize] defines how to use DCAF messaging for establishing a Datagram Transport Layer Security (DTLS) [RFC6347] channel between actors on the constrained level (cf. [I-D.ietf-ace-actors]), this document specifies a binding of DCAF to the CBOR Encoded Message Syntax, COSE [I-D.ietf-cose-msg].

To reduce confusion, we use "DTLS DCAF" to refer to DCAF based on DTLS security, and "COSE DCAF" to refer to DCAF as defined in the present document.

DCAF defines authorized access to a resource hosted on a resource Server (S) based on a security context that is established between the requesting Client (C) and S. In DTLS DCAF, this security context is tied to a DTLS channel which allows for end-to-end integrity protection and confidentiality of communication. In the presence of
intermediaries such as, e.g., CoAP proxies, channel security may not be applicable for all configurations. If this is the case, the exchanged information must be protected at the application level to help achieving the principals’ security requirements. The IETF working group CBOR Object Signing and Encryption (COSE) has defined a Concise Binary Object Representation (CBOR) [RFC7049] representation for signed and encrypted objects as well as message authentication codes.

This specification uses this CBOR Encoded Message Syntax [I-D.ietf-cose-msg] to protect the DCAF protocol flow on the application level. The features of this DCAF profile are:

- Authenticated exchange of authorization information.
- Simplified authentication on constrained nodes by handing the more sophisticated authentication work over to less-constrained devices.
- Support of secure constrained device to constrained device communication.
- Authorization policies of the principals of both participating parties are ensured.
- Simplified authorization mechanism for cases where implicit authorization is sufficient.
- Can be made to work just using symmetric encryption on the constrained nodes.
- Enable delivery of piggybacked protected content as discussed in [I-D.gerdes-ace-dcaf-authorize].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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

This specification retains the most important features of DCAF by utilizing the same basic messaging mechanism. DCAF ensures that protected information is accessible only by authorized entities, i.e. access must be authenticated and the principal that oversees the particular piece of information must permit the requested action. The DCAF specification cryptographically ties this authorization to a DTLS session setup between the communicating Client and Server. The DTLS key material used for creating this session hence defines the security context between the communicating parties. By supplementing DCAF with the notion of a context identifier, the same mechanism can be used with application level security as well.

2.1. Sending Authorized Requests

In general, every request that C sends to S must be treated by S within a particular security context with C. If S is not able to otherwise identify the security context from the message context, the context identifier must be transferred within the respective message. An example of a request containing an explicit context identifier is shown in Figure 1 using the CBOR diagnostic notation as defined in Section 6 of [RFC7049] to describe the actual data represented in CBOR.

```plaintext
PUT /r
Content-Format: application/cose+cbor
[ h'a10300',                    # protected { content_type: text/plain } 
  { alg: HMAC 256/256,          # unprotected 
    kid: h'32338647613239'    # context identifier: "238dsa29" 
  },
  h'48656c6c6f20576f726c21',  # payload: "Hello World!\n"
  h'....',                    # tag: HMAC(options+protected+payload, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]
```

Figure 1: Example for CoAP Request with Explicit Context Identifier

Figure 1 shows a PUT request from C to S for resource r containing a payload of type ‘application/cose+cbor’ that carries a COSE_Mac structure to integrity-protect the request using the MAC key from a previously established security context with identifier ‘238dsa29’. As the security context can be determined from the context identifier, an empty COSE_recipient structure is used. Note that the integrity protection not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as canonicalized external_aad as described in Section 6.
Note1: Where confidentiality is required, a COSE_encryptData structure will be used instead of the COSE_Mac structure.

Note2: COSE_enveloped may be used instead of COSE_encryptData when dynamically generated session keys should be used, e.g. with protected piggybacked content.

Note3: As transferring COSE objects as the CoAP message payload is not always possible (e.g. in GET requests), this specification defines two new CoAP options ‘Authorization’ and ‘Authorization-Format’ that can be used to convey the authorization information.

To retrieve a resource representation using the request method GET, the authorization information is conveyed in an Authorization attribute as shown in Figure 2.

GET /r

Figure 2: Example for CoAP Request with Authorization Option

The request in Figure 2 uses the default Authorization-Format ‘application-cose’ for the contents of the Authorization option which is a COSE_Mac structure. As in Figure 1 the MAC key from a previously established security context with identifier ‘238dsa29’ is used and an empty COSE_recipient structure is used. The integrity protection for this request not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as external_aad. The external_aad MUST be constructed as CBOR bytes containing a canonicalized CoAP message as specified in Section 6.

2.2. Responding to an Authorized Request

A response to an Authorized Request that uses this DCAF profile MUST be protected according to the principals’ security objectives covered by the existing security context between C and S. Usually, this means that a resource representation returned by S in the response is wrapped into a COSE_encryptData or COSE_enveloped structure. A protected response to an authorized GET request is depicted in Figure 3.
Note: For AEAD ciphers, confidentiality and integrity can be achieved in one encryption step. For other cipher suites, it may be more convenient to use a COSE_Mac structure when only message integrity is required.

2.05 Content
Content-Format: application/cose+cbor

{ h’a10300’, # protected { content_type: text/plain }
  { alg: AES-CCM-16-64-128, # unprotected
    nonce: h’77cd8a8047b7af7113bb074bccc’, # nonce
  },
  h’TBD:encrypted payload w/ tag’, # ciphertext
  # recipients:
  [ [ h’’, # protected (absent for AE alg.)
      { alg: A128KW, # unprotected
        kid: h’3233386473613239’ # context identifier: "238dsa29"
      },
      h’fec31142bc...’ # encrypted session key
    ]]
}

Figure 3: Example for a Protected Response Containing a Resource Representation

3. Establishing a Security Context

Section 2.1 illustrates the use of CBOR Encoded Message Syntax for sending Authorized Requests and Responses. Before this communication can take place the security context must be established using the COSE DCAF message types as described in this section. This section describes the basic message flow as outlined in [I-D.gerdes-ace-dcaf-authorize], but using the CBOR Encoded Message Syntax to convey the DCAF information instead of DTLS.

3.1. Unauthorized Resource Request Message

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

- The request has been received unprotected.
- The security context for the received request is unknown.
- S has no valid access ticket for the sender of the request regarding the requested action on that resource.
S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access has been granted and a security context 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 3.2. S MAY include a protected piggybacked response with the SAM Information Message in the Unauthorized Resource Request message, as discussed in Section 4.

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 naive 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.

3.2. SAM Information Message

The SAM Information Message is sent by S as a response to an Unauthorized Resource Request message (see Section 3.1) to point the sender of the Unauthorized Resource Request message to S’s SAM. The SAM information is a set of attributes containing a URI 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 4 shows an example for a SAM Information message payload using stylized CBOR diagnostic notation. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)
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.

The content format accepted by S is 'application/cose+cbor' defined in [I-D.ietf-cose-msg] to indicate DCAF over CBOR Encoded Message Syntax as defined in this document.

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 5 illustrates the binary encoding of the message payload shown in Figure 4.

```
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)
 81                        # array(2)
 19 03e7                   # unsigned(999) (=cose+cbor)
```

Figure 5: SAM Information Payload Example encoded in CBOR

3.3. 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 COSE_encryptData or COSE_enveloped structure with content-type application/dcaf+cbor 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 [I-D.gerdes-ace-dcaf-authorize].

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 as described in [I-D.gerdes-ace-dcaf-authorize]. An example Access Request from C to CAM is depicted in Figure 6. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)

Figure 6: Access Request Message Example

The example shows an Access Request message with COSE payload that contains the encrypted and integrity protected DCAF object shown in Figure 7. To integrity-protect the CoAP message header fields the canonicalized CoAP message MUST be included in the external_aad structure. The recipient structure of this message contains a wrapped key that is encrypted with the key material for the common security context of C and CAM that is identified by the kid parameter. If the client cannot create a random session key, it
could send a COSE_encryptData structure instead using the direct encryption method. The benefit of wrapping the content encryption key is that CAM can pass the encrypted content on to SAM needing to wrap the content encryption key with the key material used in the common security context with SAM.

```
{
    SAM: "coaps://sam.example.com/authorize",
    SAI: ["coaps://temp451.example.com/s/tempC", 5],
    TS: 168537
}
```

Figure 7: Access Request Payload Example

The example shows an Access Request message 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.

3.4. Ticket Request Message

CAM processes any Access Request message received from C as defined in [I-D.gerdes-ace-dcaf-authorize]. If CAM decides to send a Ticket Request message to the SAM provided in the Access Request, it has to establish a security context with SAM. 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"), or a COSE_enveloped structure using SAM’s public key to encrypt the content encryption key.

3.5. Ticket Grant Message

A Ticket Request Message is processed and responded to as specified in [I-D.gerdes-ace-dcaf-authorize]. SAM MUST use the same security context that has been used by CAM to transfer the Ticket Request message, i.e., if the Ticket Request message was received over DTLS, the response MUST be sent over the same DTLS session. This restriction is alleviated slightly when using COSE where the only requirement is that the CoAP response can be mapped to the respective request.
3.6. Ticket Transfer Message

A Ticket Transfer message is sent by CAM to deliver the authorization information from SAM in a Ticket Grant message to the requesting client C. Processing of the Ticket Grant message and construction of the Ticket Transfer message is done as specified in [I-D.gerdes-ace-dcaf-authorize]. An example for a Ticket Transfer message in response to the Ticket Access Request described in Section 3.3 is depicted in Figure 8.

2.05 Content

Content-Format: application/cose+cbor

```
{  
  h'ae1031862', # protected {  
    content_type: application/dcaf+cbor }  
  {  
    alg: AES-CCM-16-64-128 # unprotected  
    nonce: h'd259f53783993e757ec9d1d957', # nonce  
    kid: h'383261622e6161733432' # context identifier: "82ab.aas42"  
  },  
  h'TBD:encrypted payload w/ tag', # encrypted DCAF payload  
}
```

Figure 8: Example Ticket Transfer Message Encoded as COSE Message

In this example, a COSE_encryptData structure is used to avoid including a recipients structure. The kid parameter referring to the same security context that has been used for the Access Request message is included with the unprotected header of the COSE_encryptData structure. The encrypted DCAF payload contains the required ticket Face and Verifier as defined in [I-D.gerdes-ace-dcaf-authorize]. In this example, the ticket shown in Figure 9 is passed in the payload field of the COSE_encryptData structure shown in Figure 8.

```
{  
  F: {  
    SAI: [ "/s/tempC", 7 ],  
    TS: 0("2013-07-10T10:04:12.391"),  
    L: 86400,  
    G: hmac_sha256  
  },  
  V: h'f89947160c73601c7a65cb5e08812026
  6d0f0565160e3ff7d3907441cd44cc9'
  CAI: [ "/s/tempC", 1 ],  
  TS: 0("2013-07-10T10:04:12.855"),  
  L: 86400  
}
```

Figure 9: Example Ticket Transfer Message
3.7. Security Association between C and S

The information contained in a Ticket Transfer message (i.e. a ticket a Face and Client Information) can be used by C to establish a security context with S. While [I-D.gerdes-ace-dcaf-authorize] defines how to infer a DTLS pre-shared key, this specification uses the verifier as MAC key in a COSE_MAC structure as described below. This structure comprises the payload of a POST request to the authorization resource hosted by S as described in Section 7.

1. The CoAP request is protected as external_aad as described in Section 6.
2. The protected header contains the parameter content_type with the value ‘application/dcaf+cbor’.
3. The unprotected header contains the alg parameter that denotes the MAC algorithm that is used at the content level.
4. The payload field of the COSE_MAC structure contains the ticket Face encoded as canonicalized CBOR structure, and the tag field is constructed using the verifier from the Ticket Transfer message as secret, and the recipients structure is filled with empty values.

The authorization for uploading authorization tickets is tied to a key that is associated to the particular ticket Face and MUST be generated by the authorized SAM. When receiving a POST request to the auth-info resource, S generates its own version of the verifier using the information contained in Face.

The distributed key derivation method is defined as follows:

- SAM and S both generate the verifier 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 content encryption key. How SAM and S exchange $K(SAM,S)$ is not in the scope of this document. They MAY use their preshared key as $K(SAM,S)$.
- SAM MUST include a representation of the session key in the Verifier.
- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using protected channels.
- SAM MUST NOT include a representation of the Verifier in Face.
- SAM MUST NOT encrypt Face.
Once S has validated the contents of the POST request using the locally generated verifier, it creates a new resource that represents this authorization and returns the Location-Path of this new resource. This path then can be used by C to update the authorization information and MUST be used by C in the kid parameter to identify this security context as described in Section 2.1.

An example for the POST request and corresponding 2.01 response is given in Figure 10. The Location-Path returned by S is subsequently used by C as identifier for the security context tied to this authorization.

C --> S
POST /authorize
Content-Format: application/cose+cbor

S --> C
2.01 Created
Content-Format: application/cose+cbor
Location-Path: 238dsa29

Figure 10: Example POST to S’s auth-info Resource and Response

4. Piggybacked Protected Content

Piggybacked protected content was introduced in [I-D.gerdes-ace-dcaf-authorize] as a possibility to deliver an encrypted resource representation without having to maintain authorization information for the respective resource. Once a requesting client has received the piggybacked content, it needs to request authorization for accessing the protected data. To do so, it constructs an Access Request as defined in Section 3.3. 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.
5. CoAP Options Authorization and Authorization-Format

The options Authorization and Authorization-Format have the properties shown in Table 1.

<table>
<thead>
<tr>
<th>No</th>
<th>C</th>
<th>U</th>
<th>N</th>
<th>R</th>
<th>Name</th>
<th>Form</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>at</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Authorization</td>
<td>opaq</td>
<td>1-103</td>
<td>(none)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ue</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>Authorization-Format</td>
<td>uint</td>
<td>0-2</td>
<td>application/codec+cbor</td>
</tr>
</tbody>
</table>

Table 1: The Options Authorization and Authorization-Format

6. Canonicalization of the CoAP Message Header

This section describes the canonicalization of parts from the CoAP message for integrity protection. As intermediaries such as caching proxies may change certain fields in a CoAP message, only those fields are considered that must not be changed by intermediaries. The canonicalized CoAP message then serves as external_aad to the COSE MAC_structure and Enc_structure as used in this specification. The canonicalized CoAP message is constructed as follows:

```
+--------+---------+---------------------------------+----------------+
| Ver | 0 | Code | Options to protect (if any) ... |
+--------+---------+---------------------------------+----------------+
```

Figure 11: Canonicalized CoAP Message Header

As shown in Figure 11, only the version bits and the message code from the CoAP base header are relevant for integrity protection. From the list of options that a message might have, only the following options are to be included with the canonicalized message.

- If-Match
- Uri-Host
- ETag
o If-None-Match
o Observe
o Uri-Port
o Location-Path
o Uri-Path
o URI-Query
o Accept
o Location-Query
o Proxy-Uri
o Proxy-Scheme
o Size

Note: The Content-Format must be contained in the protected header of the MAC_structure or Enc_structure and hence is not required here.

An application that requires integrity protection of new options that are not listed here must add a critical-options header field to the MAC_structure or Enc_structure containing a CBOR array with the additional options to protect in ascending numerical order.

Figure 12 shows an example for a POST request to upload SenML [I-D.jennings-core-senml] sensor readings to a remote server. The protected header in the COSE_Mac structure contains a ‘required options’ entry that lists the custom option X-Something, hence the external_aad would contain a canonicalized message header that consists of the CoAP version number, the method POST, Uri-Path ‘measurements’, Uri-Path ‘current’, and X-Something 1234 as delta-encoded options in ascending order as specified in Section 3.1 of [RFC7252].
POST /measurements/current
Content-Format: application/senml+cbor
X-Something: 1234
Authorization: [
    # protected { content_type: application/senml+cbor,
    #             "required options": [ X-Something ] }
    h’a203766170706c69636174696f6e2f73656e666963696c652f70726f6a6563742f73656e6d696f6e69732f312e31", 
    { alg: HMAC 256/256, # unprotected
      kid: h’3233386473613239’ # context identifier: "238dsa29"
    },
    h’a2231a4eaec5d....’, # payload: "{ -4: 1320078..."
    h’....’, # tag: HMAC(options+protected+payload, secret)
    [ [ nil, {}, h’’ ] ] # recipients
}

{-4: 1320078429,
 -2: [{0: "temperature", 2: 272, 1: "Cel"},
      {0: "humidity", 2: 80, 1: "%RH"}]
}

Figure 12: Example Message with Protected Custom Option

7. The "auth-info" Link Relation

This section defines a resource type "auth-info" that can be used by
clients to establish a new security context with S using the
authorization information retrieved from SAM. When used with the
parameter rt in a web link, "auth-info" indicates that the
corresponding target URI can be used in a POST message to upload the
authorization information contained in the request payload.

The following example shows the web link used by S in this document
to accept authorization information created by SAM.

<authorize>;rt="auth-info";ct=TBD1,ct_cose_msg
;ct="Upload Authorization Information"

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-info";ct=TBD1,ct_cose_msg
;ep="node138"
;ct="Upload Authorization Information"
;anchor="coaps://s.example.com/"
8. Security Considerations

The SAM Information message cannot be protected as no security context between S and C is present at the time the message is sent. An attacker thus can inject a SAM Information message listing a different SAM URI to trick C into disclosing the intended action. Where this is an issue, C could retrieve the SAM URI from a resource directory as described in [I-D.gerdes-ace-dcaf-authorize].

9. 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.

9.1. CoAP Option Registration

IANA is requested to add the following entries to the CoAP Option Numbers registry:

+--------+----------------------+------------+
| Number | Name                 | Reference  |
|--------+----------------------|------------|
| 64     | Authorization         | [RFC-XXXX] |
| 65     | Authorization-Format  | [RFC-XXXX] |

10. Acknowledgements

The authors would like to thank Carsten Bormann for his valuable input and feedback.

11. References

11.1. Normative References

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-03 (work in progress), September 2015.
11.2. Informative References

[I-D.ietf-ace-actors]

[I-D.ietf-cose-msg]

[I-D.jennings-core-senml]

[RFC6347]

[RFC6838]

Editorial Comments

[_1] Editor’s note: As a consequence, if no such security context is found, the request will be rejected as Unauthorized Request.

[_3] Editor’s note: message type, token and id can change on the way.

[_4] TBD
Authors’ Addresses

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

Phone: +49-421-218-63904  
Email: bergmann@tzi.org

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

Phone: +49-421-218-63906  
Email: gerdes@tzi.org
Privacy-Enhanced Tokens for Authorization in ACE
draft-cuellar-ace-pat-priv-enhanced-authz-tokens-01

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on April 19, 2009.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Abstract

This specification defines PAT, "Privacy-Enhanced-Authorization-Tokens" or "Pseudonym-based Authorization Tokens", a protocol and a token construction procedure for client authorization in a constrained environment, similar to DCAF [I-D.gerdes-ace-dcaf-authorize].

The tokens can be also used to establish a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes.

Table of Contents

1. Introduction...................................................3
   1.1. Key words to Indicate Requirement Levels.................3
   1.2. Features..................................................4
   1.3. Actors and Terminology....................................5
2. System Overview................................................6
3. Protocol Overview..............................................7
   3.1. Message Flow Overview....................................8
4. Protocol Details..............................................10
   4.1. [C --> S: Unauthorized Resource Request: request_params].10
   4.2. [S --> SAM: Token Request]...............................10
   4.3. [SAM --> S: Token Response: ST = (St, paramS)]...........11
   4.4. [S --> C: SAM Information: SAM-ID]..........................11
   4.5. C --> CAM: Access Request................................11
   4.6. CAM --> SAM: Ticket Request...............................11
   4.7. SAM --> CAM: Ticket Grant...............................11
   4.8. CAM --> C: Ticket Transfer...............................12
   4.9. CAM --> C: Client Authorization Information.............12
   4.10. C <==> S: Security Association..........................12
   4.11. C --> S: Authorized Resource Req.: AT = (At, param),[params].................................................12
5. Construction of the Tokens....................................13
   5.1. Main data structure....................................14
      5.1.1. Traversing the Tree................................15
   5.2. Construction of St, Ct and At............................15
6. Formal Syntax..................................................16
7. Security Considerations.......................................16
8. IANA Considerations...........................................16
9. Conclusions...................................................16
10. References..................................................16
1. Introduction

Three well-known problems in constrained environments are the authorization of clients to access resources on servers, the realization of secure communication between nodes, and the preservation of privacy. The reader is referred for instance to [I-D.gerdes-ace-dcaf-authorize] and [I-D.gerdes-ace-actors], and [KoMa2014].

This draft tackles certain aspects of those three problems. It describes a way of constructing Token Material (Key Material) that can be used by clients and servers (or in some cases, more generally by arbitrary nodes) to create secure channels, provide authentication, in a context similar to ACE-DCAF Tickets. Moreover, the construction can be used to offer user consent (in the sense of privacy) and to create dynamically pseudonyms to enhance the unlinkability of the information, see Subsection "Features" below.

This draft uses the same architecture of [I-D.gerdes-ace-actors], designed to help constrained nodes with authorization-related tasks via less-constrained nodes. As in DCAF, PAT supports an implicit authorization mode where no authorization information is exchanged and 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.

A main goal of PAT is to securely transmit authorization tokens. A by-product is the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] between two nodes. Notice that the DTLS channel is not needed to securely transmit the authorization tokens. In some cases, relevant in constrained environments, it is also not necessary for a secure transmission of the payload data from server to client.

1.1. Key words to Indicate Requirement Levels

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].
In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

1.2. Features

- The method allows a User, or an Authentication/Authorization Manager on its behalf, to authorize one (or several) client(s) to access resources on a server. The client and/or the server can be constrained devices. The authorization is implemented by distributing purpose-built Key Material (which we generically call "Tokens") to the server and clients. This SHOULD be done by secure channels.

- The Client Tokens are crafted in such a way that the clients can construct authorization tokens that allow them to demonstrate to the server their authorization claims. The message exchange between client and server for the presentation of the tokens MAY be performed via insecure channels.

- Further, the purpose-built Key material and tokens can be used for establishing a secret shared key between a client and the server, which can be then used to establish a DTLS communication with pre-shared keys.

- The tokens do not provide any information about any associated identities or identifiers of the clients nor of the server. In particular, the method can be used in context where unlinkability (privacy) is a main goal: the tokens convey only the assurance of the authorization claims of the clients.

This means that the payloads of our protocol, and in particular, the Authentication Token secrets used, can be constructed in such a way that they not leak information about the correspondence of messages to the same Client. In other words: if an eavesdropper observes the messages from the different Clients to and from the server, the protocol does not give him information about which messages correspond to the same Client. Of course, other information, like the IP-addresses or the contents themselves of the requests/responses may leak some information in this regard, but that is not information leaked by our protocol and can be treated separately.
The tokens may be supported by a "proof-of-possession" (PoP) method. PoP allows an authorized entity (a client) to prove to the verifier (here, the server), that he is indeed the intended authorized owner of the token and not simply the bearer of the token. (Notice that the Authorization Token may be sent in the clear, and thus, it could be stolen by an intruder. A PoP would hinder the attacker to use the token pretending to be authorized).

- The Key Material can be used to generate and coordinate pseudonyms between C and S and potentially further parties.

- The user (more precisely, the Resource Owner, RO, see Section "Actors and Terminology" below) is able to decide (if he wishes: in a fine-grained way and in real-time) which client under which circumstances may access his data stored in S. This can be used to provide consent (in terms of privacy) from users (again, ROs).

As DCAF, it has the following features:

- Simplified authentication on constrained nodes by handing the more sophisticated authentication over to less-constrained devices.

- Support of secure communication between constrained devices.

- Authorization policies of the principals of both participating parties are ensured.

- Simplified authorization mechanism for cases where implicit authorization is sufficient.

- Using only symmetric encryption on constrained nodes.

1.3. Actors and Terminology

The actors and terminology are the same as in DCAF. Very briefly, for the purposes of this draft, the main actors are:

Server (S): An endpoint that hosts and represents a CoAP (see [RFC7252]) 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.

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

In order to avoid confusions, instead of redefining the terms of DCAF, we use additionally the following terms:

Server Token (ST): The token which is generated by the SAM for the Server. Besides parameters, which may contain authorization information that represents RO’s authorization policies for C, it contains a secret, St, called the ST-secret. This one can be used to verify the Authorization Token and to generate other secrets to be discussed later.

Client Token (CT): The token which is generated by the SAM for the Client. It contains a secret, Ct, which can be used to generate the Authorization Token, plus some other data used for PoP. Optionally CT may contain authorization information that represents RO’s authorization policies for C.

Authorization Token (AT): The token which is generated by the Client and presented by him to the Server. It contains a secret AT, which changes regularly (in a similar way to one-time passwords). The AT contains all information needed by the Server to verify that it was granted by SAM.

VerifK, PSK, IntK, ConfK: Derived keys between C and S used respectively:
- to verify that they are talking with the intended partner, for the Client C it is used as Proof of Possession of the (current) Authorization Token
- as Pre-shared Key to establish a DTLS secure channel
- for Integrity protection (in message authentication codes)
- for Confidentiality Protection (to be elaborated in a future version of the document).

2. System Overview

As in DCAF, each Server (S) has a Server Authorization Manager (SAM) which conducts the authentication and authorization for S. S and SAM are assumed to have a secure channel, probably a DTLS channel, but the current specification does not assume anything about it, except that it is two way secure, preserving integrity and confidentiality. Using this secure communication channel SAM provides to S the main
secret x which is used within the initial version of the Server Token (ST). Thus ST = (St, paramS), where St is a "main secret" created by SAM (in a way that is outside of the scope of this draft), and paramS is a set of parameters, determining the functions G, g1, g2, g3, g4, etc. to be discussed later and, optionally, the authorization policies for the clients foreseen.

To gain access to a specific resource on a Server S, a Client (C) requests a token from the SAM, either directly or using its CAM. In the following, for simplicity, we only discuss the collocated CAM-C role; the separation of the roles should be clear to the reader (and will be detailed in subsequent versions of the ID).

After SAM receives the request from C, he decides if C is allowed to access the resource. If so, it generates a Client-Id and a corresponding Client-Token used for the authorization and for securing the communication between C and S.

For explicit access control, SAM adds the detailed access permissions to the token in a way that C (or his CAM) can interpret and S can verify as authentically stemming from SAM.

Then C presents the Authorization Token to S, demonstrating his authorization, and C and S can establish a secure channel.

As in DCAF, an Authorization Manager has to fulfill several requirements regarding enough storage, use interaction and processing power, see [I-D.gerdes-ace-a2a].

3. Protocol Overview

The PAT protocol is an instantiation of the generic ace-solution protocol presented in [I-D.cuellar-ace-solutions]. It comprises the following steps:

(0) Security Context Setup Between CAM and SAM
(1) Unauthorized Resource Request Message
(2) Token Request Message
(3) Token Response
(4) SAM Information Message
(5) Access Request Message
(6) Ticket Request Message
(7) Ticket Grant Message
(8) Ticket Transfer Message
(9) Client Authorization Information Message
(10) Security Association between C and S
(11) Authorized Resource Request Message
(12) Resource Response Message

PAT instantiates in particular the payload of messages (3), (7), (8) and (11). The main exchanges can be presented as in Fig. 1:

1. Transfer of ST = (St, paramS) form SAM to S. This corresponds to the message (3) Token Response of [I-D.cuellar-ace-solutions]

2. The Client Token (CT) grant from SAM to C. This is the composition of messages (7) and (8).

3. Access Requests with their respective Authorization Tokens (AT) between C and S.

Figure 1: The 3 main parts of the Protocol

There are 3 main Tokens: ST, CT and AT, each of the form (nonce, param), where the nonce is St, Ct, and At, resp., and param = paramC/paramS/paramT is some additional information. (ParamT is not shown in the Figure, for readability).

3.1. Message Flow Overview

In Figure 2, a PAT protocol flow is depicted (messages in square brackets are optional). Notice that in comparison to DCAF, rows 07 and 08 are in different order and the DTLS channel between C and S is optional. The resource response (09) can be optionally secured by DTLS or by other native PAT methods:
As in DCAF, 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. Or, 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 the available resources.

Once C knows SAM’s address, it can send a request for authorization to SAM (directly, as in Fig. 1 or indirectly using its own CAM). If the access is to be authorized, SAM generates a Client Token (CT) for C. It contains keying material for generating all necessary tokens and keys, and, if necessary, a representation of the permissions C has for the resource.
Each time C sends S a Resource Request, it generates and presents a (current) Authorization Token to S to prove its right to access. With their common knowledge in St and Ct, C and S are able to establish a secure channel.

The following sections specify the message flows in more detail, how the token secrets St, Ct and At are constructed, how the tokens can be revoked, and how S and C can use their common knowledge to verify the authenticity of the ATs and to obtain the shared keys VerifK, PSK, IntK, and ConfK.

4. Protocol Details

In the following descriptions the notation

\(<\text{Msg\_Name}> \ A \rightarrow B : \text{payload}>\)

represents the message with name \text{Msg\_Name}, sent from A to B and with the given payload.

We assume that the Server S and its Authentication Manager SAM share a secure channel, which may be implemented via USB (and physical security) or via DTLS, etc. We do not assume any particular concrete secure channel, but it must be stressed that the security of the protocol strongly depends on how this security is designed and implemented.

We also assume that the CAM and the SAM share a secure connection, say over DTLS.

4.1. \([C \rightarrow S: \text{Unauthorized Resource Request: request\_params}]\)

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 two following holds:

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

4.2. \([S \rightarrow \text{SAM: Token Request}]\)

Optionally, the server may ask for server tokens to the SAM.
4.3. [SAM --> S: Token Response: ST = (St, paramS)]

The owner of the server determines a number N which is (probably) an upper bound on the number of Clients that the Server will simultaneously serve. This number N should not be too high, as the storage and computation effort of the server will increase (linearly) with N. (But the owner may decide any time later to increase or decrease the number N if necessary). Using the secure channel, SAM sends to S the initial value of ST = (St, paramS), where St is a (preferably, random) number that can't be guessed by an attacker, and paramS is a set of parameters that encode the number N, the choice of functions G, g1, g2, g3, g4, and the permissions Client Nr "i" has (for each Client i, or for a set of them). The permissions may remain undefined or incomplete and can be extended or modified later anytime. They may also contain validity periods or other restrictions in the Service Level Agreement.

At any later point in time the SAM may change ST, or a part of it: send a new value for St, or change or extend the permissions or change N, the number of expected Clients.

4.4. [S --> C: SAM Information: SAM-ID]

As in DCAF, the Server CAN instruct the Client about which SAM to contact.

4.5. C --> CAM: Access Request

The Client contacts directly or indirectly via the CAM the SAM of his desired Server S, expressing the set of permissions it requests to the resources of the Server S.

4.6. CAM --> SAM: Ticket Request

The Client contacts directly or indirectly via the CAM the SAM of his desired Server S, expressing the set of permissions it requests to the resources of the Server S.

4.7. SAM --> CAM: Ticket Grant

SAM decides which Client Number "i" the Client C should have. Each Client will have a different number. The number "i" is an integer between 1 and N, the number of Clients. The choice of value for "i" will depend on which permissions the owner has foreseen and, more importantly, the SAM has encoded as parameters sent to S.
In this message the Client Token CT and the number i are sent.

Optionally, SAM can encode the permissions in this message in a way that the Server S can verify the authenticity of the permission. (Details will be given in a later version of the draft).

4.8. CAM --> C: Ticket Transfer

The CAM forwards the Client Token to the client C.

4.9. CAM --> C: Client Authorization Information (CAI)

The CAM may restrict the operations C performs on S by transferring Client Authentication Information (CAI) to C.

4.10. C <==> S: Security Association

Optionally, a DTLS channel is constructed using pre-shared key constructed from the common information held by C and S.

4.11. C --> S: Authorized Resource Req.: AT = (At, param),[params]

In possession of the Client Token, CT, the Client can construct valid Authorization Tokens, AT, which demonstrates his authorization to access the resources he is requesting.

Regularly, the message Resource_Req has to be sent afresh and a new AT must be used: Client C has to renew his Authorization status at the Server. The frequency in which the Client has to send a new AT can be enforced by C and is determined indirectly the owner of S (or by SAM). This allows a fine-grained control on the service level that the Server will provide to the Client (for instance, on the amount of information of sensor data). We assume that the frequency of renewal is the same for all Clients, but each Client has a different number of Authorization Tokens it can construct.

Each time a new Resource_Req is sent, a new Authorization Token MAY be needed.

4.12. S --> C: Resource Response

The server answers the request of the server, as stipulated by the service description. This message can be sent over the secure channel between C and S (established as described above), or can be secured directly, as discussed below.
5. Construction of the Tokens

The main data structure used in this document may be represented as a tree of values. Each value is a bit string of a fixed size, which we denote m. Initially we choose m = 265 bits, but it is easy to define extensions for other values of m. This data structure may be implemented in several different ways, for instance as a set of tables representing the currently relevant parts of the tree.

The tree is constructed based on a "root secret" which we denote by "x") and a Pseudo-Random Generator, commonly used to generate Stream Ciphers. In particular, we propose to use the Pseudo-Random Generator (PRG) of ChaCha20 [RFC7539]. In other words, we use ChaCha20 block function as a key-derivation function, by generating an arbitrarily long keystream. The stream cipher ChaCha20 takes as input a 256-bit key k, a 64-bit nonce v (unique message number), and a 64-bit block number. The ChaCha20 output stream can therefore be accessed randomly, and any number of blocks can be computed in parallel.

Instead of ChaCha20, other PRG can be used as well, or even hash functions. With any of those building blocks, it is easy to construct functions G: K x I -> K, and g2, g3, g4: K x I -> K, where K = the key space = \{0,1\}^265 and I ={0,1,2,..N} where N is an appropriate integer (a parameter of the construction).

Starting from a secret x, a tree of derived secrets (we use the words keys and secrets indiscriminately) is constructed. The main property of the secrets in the tree is that an attacker can’t use the information of a secret to obtain information about other secrets in the tree except descendents. (The knowledge of secrets on the tree reveals nothing about any secret that is not a descendent of any of them).

The children of any node are constructed using a function G ("generator") that takes a key k (of size m = 256 bits) and an index i (the "block number") and creates a new key of size m =256 bits.

The Token secrets St, Ct and At are all values in the tree and thus can be constructed from x using G. Other functions g1, g2, g3, and g4 will be used to generate the derived keys VerifK, PSK, IntK, and ConfK.

We assume that G, g1, g2, g3, and g4 are (or may be) all publicly known functions. That is the security of the protocol SHOULD NOT depend on the secrecy of those functions.
5.1. Main data structure

The main data structure used in this document may be viewed abstractly as a tree of values. Each value is a bit string of a fixed size m (= 256 bits). But this data structure may be implemented in several different ways, for instance as a set of tables representing the currently relevant parts of the tree.

We use a sequence of integer numbers as indexes for the nodes (values) in the tree. To avoid parentheses, commas, and semicolons we write for instance: "123" for the sequence of 3 numbers "1", "2", and "3". In what follows, in all our examples of integer sequences, we will not use numbers that require 2 or more digits (that is, numbers > 9).

The sequences of integers are used to index values in a tree: x_a is the value at the node with position (address) a. In other words, the nodes (and their values) are denoted as x_a, where a is a sequence of integer numbers. The tree has a root x (x_a where a = epsilon is the empty sequence). The children of x are x_1, x_2, x_3, ..., x_N, where k = 1..N is a singleton list, that is a list with only one number. If x_a is a node in the tree, then the children of x_a all have the form x_a', where a'=a;i is a the concatenation of a and an integer i. The value x_a' = x_(a;i) is calculated as G(x_a, i). In the case of a hash h:

\[ x_{a'} = x_a;i = h(f(x_a, i)). \]

Where f is a fixed (publicly known) function such that for any fixed i the function f(.,i) is 1-1. The choice of G should not be regarded a secret: it is a publicly known parameter of the installation for S. It follows that if an entity knows x_a, the entity is able to calculate all descendants of it, that is, all nodes in the subtree with root x_a. But not vice-versa: since h is a one-way function, the knowledge of x_a;i is not enough to calculate x_a.

Note: "x_a" is read as "x sub a" or "x subindex a". "a" is called the index or address of the node.

Note: Since we also use concatenation of bit strings we need to use parenthesis in that case: x_a;i means x_(a;i), while (x_a);bs means the concatenation of the bit strings (x_a) and bs.
5.1.1. Traversing the Tree

Now we describe a simple procedure for traversing part of the tree, which we now assume of a fixed degree. That is, each node of the tree has either no children or has exactly this amount of children.

Assume that we have a certain "current parent node" $x_a$ and a "current node", a child of $x_a$, which may be written as $x_a;i$. Thus, $i$ is an index (an integer). For example: the fifth child of $x_a$ is $x_a;5$, with $i = 5$.

Traversing the tree with respect to the current parent node $x_a$, starting at $x_a; i$ gives the following sequence of nodes (loop):

- If $x_a;i$ is the right-most child of $x_a$, stop here.
- If $x_a;i+1$ is the right-most child of $x_a$, then move to its first child (if it exists): $x_a;i+1;1$. If $x_a;i'+1$ has no children, stop.
- If $x_a;i'+1$ is not right-most child of $x_a$, then move to it.

5.2. Construction of St, Ct and At

The secret "x" is the "main secret". It is generated by the SAM as a random or pseudo-random number of $m$ bits ($m$ is taken to be 256). The method used to construct $x$ is out of our scope, but it should be practically impossible to guess by an attacker, even if he knows in plaintext a sequence of previous or future choices of $x$.

Initially, $St = x$

$St$ is sent by the SAM to the Server $S$ in the message $Server_Token_Transfer$. The value of $St$ at the Server may change if the current value of $St$ is revoked by the SAM. For this, it is not necessary to send a new $Server_Token_Transfer$. (Details in a future version of the I.D).

The root has $N$ children, one for each foreseen Client. The value $x_i$, for $i=1..N$ is a secret associated to Client number $i$, but it is not known by the Client. The values $x$, $x_1$, $x_2$, ..., $x_N$ are secrets that never leave the SAM or the Server $S$ and should not be leaked.

The first children of $x_1$, $x_2$, ..., $x_N$ are the initial values of $Ct$. In other words, for Client number $i$:

Initially, $Ct = x_i1$
Ct is sent by the SAM to C(i), the Client number i, in the message Client_Token_Grant. Also in this message, the SAM sends the "current node" (used by C to start a traversal), the depth and the degree of the sub-tree at the node Ct.

The value of Ct at the Server may change if the current value of Ct is revoked by the SAM. If this happens, it is necessary for the Client to obtain a new Client_Token_Grant. (Details in a future version of the I.D).

To create the sequence of Authentication Token secrets, At1, At2, ..., the Client traverses the tree starting at a "current node" (determined by the SAM in the parameters of the Client_Token_Grant message) with the current parent node being the current value of Ct.

6. Formal Syntax

tbd

7. Security Considerations

tbd

8. IANA Considerations

tbd.

9. Conclusions

tbd

10. References

10.1. Normative References


Informative References


11. Acknowledgments

This draft is the result of collaborative research in the RERUM EU funded project and has been partly funded by the European Commission (Contract No. 609094).

This document was prepared using 2-Word-v2.0.template.dot.
Appendix A.

A.1. Copyright Statement

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info).
Authors’ Addresses

Jorge Cuellar
Siemens AG
CT RTC ITS

Email: jorge.cuellar @ siemens.com
Abstract

The Constrained Application Protocol (CoAP) is a transfer protocol that was designed to meet the special requirements of constrained environments.

This document introduces a common framework for conveying authorization information between the actors in the ACE architecture by defining classes of message types. It thus specifies a common authorization extension for CoAP.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document.
Table of Contents

1. Introduction ................................................. 3
   1.1. Terminology ............................................ 4
2. Overview ...................................................... 4
3. Message Type Classes ........................................... 4
   3.1. Unauthorized Resource Request Message .................. 5
      3.1.1. Effect .............................................. 5
      3.1.2. Actors .............................................. 5
      3.1.3. Protection Requirements ............................ 5
   3.2. SAM Information Message ................................ 5
      3.2.1. Effect .............................................. 5
      3.2.2. Actors .............................................. 6
      3.2.3. Protection Requirements ............................ 6
   3.3. CAM Information Message ................................ 6
      3.3.1. Effect .............................................. 6
      3.3.2. Actors .............................................. 6
      3.3.3. Protection Requirements ............................ 6
   3.4. Access Request Message .................................. 6
      3.4.1. Effect .............................................. 6
      3.4.2. Actors .............................................. 6
      3.4.3. Protection Requirements ............................ 7
   3.5. Ticket Request Message .................................. 7
      3.5.1. Effect .............................................. 7
      3.5.2. Actors .............................................. 7
      3.5.3. Protection Requirements ............................ 7
   3.6. Ticket Grant Message .................................... 7
      3.6.1. Effect .............................................. 7
      3.6.2. Actors .............................................. 8
      3.6.3. Protection Requirements ............................ 8
   3.7. Ticket Transfer Message .................................. 8
      3.7.1. Effect .............................................. 8
      3.7.2. Actors .............................................. 8
      3.7.3. Protection Requirements ............................ 8
   3.8. Client Authorization Information Message ............... 8
      3.8.1. Effect .............................................. 8
      3.8.2. Actors .............................................. 9
      3.8.3. Protection Requirements ............................ 9
   3.9. Security Context Setup Between CAM and SAM ............. 9
      3.9.1. Effect .............................................. 9
      3.9.2. Actors .............................................. 9
Internet-Draft ace-solutions October 2015

1. Introduction

Resource-constrained nodes 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. They are often especially designed to perform a single, simple task in their application area. The various use cases (see [I-D.ietf-ace-usecases]) have varying requirements for the authentication and authorization solution. Due to the constrainedness of the devices, a single solution cannot address all these requirements.

In the Authentication and Authorization for Constrained Environments (ACE) working group, various proposals are discussed that cover different use-cases and application scenarios. This document explains how the specific solutions in the ACE WG fit together in a common framework. It defines classes of message types to convey authenticated authorization information between the actors in the ACE architecture. [I-D.ietf-ace-actors]

The description of each message type covers the effect this message has, the actors that send and receive the message and the kind of protection it requires. Solution designer can implement the message type classes with the effect they require for their solution.
1.1. Terminology

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

2. Overview

The ACE architecture as outlined in [I-D.ietf-ace-actors] introduces six actors – logical entities that have to perform specific security-related tasks; On the constrained level, client and server want to communicate securely. Their respective principals define authorization policies that need to be enacted. Each constrained device has a less-constrained device that can be entrusted with security-related tasks. One goal of the ACE WG is to enable entities on the constrained level to securely delegate some authorization-related tasks to an actor on the less-constrained level within the same security domain.

The ACE architecture facilitates various distinct application scenarios resulting in the following basic authorization message flows.

1. To access a resource on a server, the client presents an authorization token together with the request.

2. When a client tries to access a resource on a server, the server retrieves authorization information for this action.

3. The server disseminates encrypted data where the decryption key is bound to the client’s authorization.

In all cases, the authorization policies of both the client’s principal and the server’s principal must be considered to achieve their respective security goals. Depending on the selected authorization message flow, different actors need to exchange different information.

This document is structured as follows: Section 3 specifies 11 classes of Message Types that define how this information is securely conveyed over the network. Section 4 describes CoAP content formats that can be used to control the desired authorization message flow.

3. Message Type Classes

In the following, the classes of message types for authorization are listed. Each class consists of the effect this message has, the actors that send and receive this message, and the kind of protection...
that such a message requires. Solutions can choose the message types they need to implement based on the effects they require.

3.1. Unauthorized Resource Request Message

Any resource request from C to S that is not covered by a valid ticket for C is treated as unauthorized request. If S decides to act upon an Unauthorized Resource Request it can reject the message and optionally inform C where it can ask for authorization, or, if S has authenticated C, S can directly ask SAM to authorize C’s request.

3.1.1. Effect

- S can act on the unauthorized request to determine if C is authorized, either by requesting authorization from SAM or by rejecting the request and optionally inform C about which SAM to contact in order to retrieve a valid authorization token.
- If S happens to be a gateway (GW) that serves content on behalf of another entity (called "origin server"), GW can act as previously described for S.

3.1.2. Actors

- C -> S
- or, optionally,
- C -> GW

3.1.3. Protection Requirements

None.

3.2. SAM Information Message

A SAM Information Message can be used by S or a gateway (GW) that serves the requested resource on behalf of an origin server S to instruct C where it may retrieve authorization for a specific type of request. S (or GW, respectively) may optionally include requested data as an encrypted object with the SAM Information Message.

3.2.1. Effect

- C knows the address of SAM (where to request a ticket for S).
3.2.2. Actors
   o S -> C
   or optionally,
   o GW -> C

3.2.3. Protection Requirements
   If S/GW includes requested data with the SAM Information Message, it must provide for confidentiality and integrity of the data.

3.3. CAM Information Message
   A CAM Information Message can be used by C to instruct S where it may retrieve an authorization token for C.

3.3.1. Effect
   o S knows the address of CAM (where to request a ticket for C).

3.3.2. Actors
   o C -> S

3.3.3. Protection Requirements
   None.

3.4. Access Request Message
   An Access Request Message is sent by C to request CAM to retrieve authorization information for a specific request. It includes information from a SAM Information message generated by S/GW.

3.4.1. Effect
   o CAM knows the resources and actions C is requesting.
   o CAM knows which SAM to contact.

3.4.2. Actors
   o C -> CAM
3.4.3. Protection Requirements

- Integrity and Authenticity (CAM can validate that the message stems from C)
- Confidentiality (optional): the principals may not want others to know which resources and actions where requested.

3.5. Ticket Request Message

A Ticket Request message is sent by CAM on behalf of C to retrieve authorization from SAM for a specific action on S.

3.5.1. Effect

- SAM knows which actions on which resources are requested by CAM.
- SAM can determine permissions for CAM.
- SAM can generate an access ticket for C, which can be later used by C to demonstrate to S its authorization status.
- SAM can generate a verifier for C, which can be later used by C to verify that it is communicating with an appropriate S.

3.5.2. Actors

- CAM -> SAM

3.5.3. Protection Requirements

- Integrity and authenticity (SAM can validate that the message stems from CAM)
- Confidentiality (optional): the principals may not want others to know which resources and actions where requested.

3.6. Ticket Grant Message

A Ticket Grant message is sent by SAM to CAM to convey authorization information and a verifier that can be used by C to access protected resources on S.

3.6.1. Effect

- CAM received the Server Authorization Information (SAI)
- CAM received the verifier for C
CAM can validate the origin of the ticket for C

3.6.2. Actors

- SAM -> CAM

3.6.3. Protection Requirements

- Confidentiality (SAM, CAM) (+ Integrity (implicit, the ticket already is integrity-protected))
- SAM knows the principal’s authorization policies for CAM

3.7. Ticket Transfer Message

The Ticket Transfer message is used by CAM to convey the authorization information and the verifier retrieved from SAM to C.

3.7.1. Effect

- C is able to prove its authorization status to S
- C is able to communicate securely with S

3.7.2. Actors

- CAM -> C

3.7.3. Protection Requirements

- Confidentiality (CAM, C) (+ Integrity if the ticket not already is integrity-protected)

3.8. Client Authorization Information Message

CAM can restrict the operations C performs on S by transferring Client Authentication Information (CAI) to C. This is specifically useful if S has requested additional information from C in order to proceed with C’s initial request.

3.8.1. Effect

- C gets the client authorization information (CAI) received from CAM
- C knows which information it is allowed to provide to S
3.8.2. Actors

- CAM -> C

3.8.3. Protection Requirements

- Integrity: attackers must not be able to manipulate the CAI.
- Confidentiality (optional): in some cases, principals might not want others to gain knowledge of the CAI.
- CAM knows the principal’s authorization policies for C.

3.9. Security Context Setup Between CAM and SAM

In the ACE architecture, the client may utilize an authorization manager (CAM) to contact the server-side authorization manager (SAM) and retrieve an authorization token for the intended action on a resource that SAM is responsible for. CAM needs to authenticate with SAM on behalf of C and must authenticate SAM. The message exchange between CAM and SAM establishes a security context that can be used to request authorization for CAM and transfer authorization policies for SAM.

3.9.1. Effect

- Mutual authentication (TODO: split)
- CAM can authenticate messages from SAM
- SAM can authenticate messages from CAM
- SAM can determine authorization policies for CAM
- CAM can determine authorization policies for SAM

3.9.2. Actors

- CAM <-> SAM

3.9.3. Protection Requirements

None.
3.10. Security Association between C and S

Once C has been authorized by SAM to access resources on S and by CAM to transmit data to S, both actors have a common security context that can be used to exchange further messages. The authorization information bound to this security context can be updated subsequently over a suitable interface provided by C and S.

3.10.1. Effect

- C can authenticate messages from S
- S can authenticate messages from C
- Further communication between C and S can be encrypted
- S knows the SAI for C

3.10.2. Actors

- C, S

3.10.3. Protection Requirements

- Integrity: Attackers must not be able to update the authorization information stored at S and C.
- Confidentiality (optional): Usually, only entities that are authorized to update the authorization information should be able to read that data.

3.11. Authorized Resource Request Message

Within the security association between C and S, request messages covered by the authorization information that is bound to the common security context are Authorized Resource Request messages that the receiver is allowed to process.

3.11.1. Effect

- S can process requests from C, C can process requests from S.

3.11.2. Actors

- C -> S
3.11.3. Protection Requirements

- Integrity
- Confidentiality (optional): the principals might not want others to know the requested resource.

3.12. Resource Response Message

Responses to Authorized Request messages are Resource Responses.

3.12.1. Effect

- C receives the requested service from S.

3.12.2. Actors

- S -> C

3.12.3. Protection Requirements

- Integrity
- Confidentiality (optional): the principals might not want others to know the response content.

3.13. Server-Initiated Ticket Request Messages

TODO (see [I-D.gerdes-ace-dcaf-sitr])

4. Content Format

As the ACE working group aims at an authorization solution that follows a REST architecture style, the basic message flow is controlled by the content format that is used to convey authorization-specific data. For example, S might transfer the SAM Information message in content format ‘application/cose+cbor’ to indicate its capability of handling messages that use the COSE message syntax [I-D.ietf-cose-msg], or ‘application/dcaf+cbor’ to use the DCAF messaging format specified in [I-D.gerdes-ace-dcaf-authorize].

5. Security Considerations

TBD
6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgements

The authors would like to thank Carsten Bormann for his valuable input and feedback.

8. Informative References

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-03 (work in progress), September 2015.

[I-D.gerdes-ace-dcaf-sitr]

[I-D.ietf-ace-actors]

[I-D.ietf-ace-usecases]

[I-D.ietf-cose-msg]

Authors’ Addresses

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

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

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

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

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ................................. 3
   1.1. Features .................................. 4
   1.2. Terminology ............................... 4
       1.2.1. Actors ................................. 4
       1.2.2. Other Terms ......................... 5
2. System Overview ............................... 6
3. Protocol ..................................... 7
   3.1. Overview .................................. 7
   3.2. Unauthorized Resource Request Message .......... 8
   3.3. SAM Information Message .................... 9
       3.3.1. Piggybacked Protected Content .......... 10
   3.4. Access Request ............................ 11
   3.5. Ticket Request Message .................... 12
   3.6. Ticket Grant Message ...................... 13
   3.7. Ticket Transfer Message .................... 15
   3.8. DTLS Channel Setup Between C and S ........... 16
   3.9. Authorized Resource Request Message .......... 17
   3.10. Dynamic Update of Authorization Information ..... 18
       3.10.1. Handling of Ticket Transfer Messages ..... 19
4. Ticket ...................................... 20
   4.1. Face ................................... 20
   4.2. Client Information ........................ 21
   4.3. Revocation ................................ 22
   4.4. Lifetime .................................. 22
       4.4.1. Revocation Messages .................. 22
5. Payload Format and Encoding (application/dcaf+cbor) ..... 23
   5.1. Examples .................................. 26
6. DTLS PSK Generation Methods ...................... 28
   6.1. DTLS PSK Transfer .......................... 28
   6.2. Distributed Key Derivation .................. 28
7. Authorization Configuration ...................... 29
8. Trust Relationships ............................ 29
   9.1. The "auth-request" Link Relation ............. 31
10. Examples ................................... 31
   10.1. Access Granted ............................ 31
   10.2. Access Denied ............................. 33
   10.3. Access Restricted ......................... 34
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

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

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Readers are expected to be familiar with the terms and concepts defined in [I-D.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 Manager (SAM) which conducts the authentication and authorization for S. S and SAM share a symmetric key which has to be exchanged initially to provide for a secure channel. The mechanism used for this is not in the scope of this document.

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

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

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

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

- AM must possess means for user interaction, for example directly or indirectly connected input/output devices such as keyboard and
display, to allow for configuration of authorization information by the respective Principal.

- AM must have enough processing power to handle the authorization requests for all constrained devices it is responsible for.

3. Protocol

The DCAF protocol comprises three parts:

1. transfer of authentication and, if necessary, authorization information between C and S;

2. transfer of access requests and the respective ticket 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. ==>   | [Resource Req.--->] | <== DTLS chan. ==> |
|                    | [-- SAM Info.]     |                    |
|-- Access Req.      |                    |                    |
<==== TLS/DTLS channel (CAM/SAM Mutual Authentication) ====>
<table>
<thead>
<tr>
<th>Ticket Request</th>
<th>-------------------------------</th>
<th>Ticket Grant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ticket Transf. --&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;== DTLS chan. ==&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auth. Res. Req. -&gt;</td>
<td></td>
</tr>
</tbody>
</table>
```

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:

- The request has been received on an unprotected channel.
- S has no valid access ticket for the sender of the request regarding the requested action on that resource.
\begin{itemize}
  \item S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.
\end{itemize}

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.)
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.

```
02  # 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) (=dacf+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'f89947160c73601c7a65cb5e088120266d0f0565160e3ff7d3907441cdf44cc9' }

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 authorization 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
       6d0f0565160e3ff7d3907441c9cc9f'
    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.
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:

- SAM Authorization Information (SAI)
- A nonce

Optionally, Face MAY also contain:

- A lifetime (optional)
- A DTLS pre-shared key (optional)
S MUST verify the integrity of Face, i.e. the information contained in Face stems from SAM and was not manipulated by anyone else. 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 revoking 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 revoke 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.
### Table 1: DCAF field identifiers encoded in CBOR

<table>
<thead>
<tr>
<th>Encoded Value</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SAM</td>
</tr>
<tr>
<td>1</td>
<td>SAI</td>
</tr>
<tr>
<td>2</td>
<td>CAI</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>K</td>
</tr>
<tr>
<td>5</td>
<td>TS</td>
</tr>
<tr>
<td>6</td>
<td>L</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
</tr>
<tr>
<td>9</td>
<td>V</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>D</td>
</tr>
<tr>
<td>12</td>
<td>N</td>
</tr>
</tbody>
</table>

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

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

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

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
Internet-Draft                    DCAF                      October 2015

CBOR are defined in Table 2. A registry for these codes is defined in Section 13.1.

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

V: Ticket Verifier. A binary string containing the shared secret between C and S.

+---------------+-------------+-----------+
<table>
<thead>
<tr>
<th>Encoded Value</th>
<th>Mnemonic</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hmac_sha256</td>
<td>mandatory</td>
</tr>
<tr>
<td>1</td>
<td>hmac_sha384</td>
<td>optional</td>
</tr>
<tr>
<td>2</td>
<td>hmac_sha512</td>
<td>optional</td>
</tr>
</tbody>
</table>
+---------------+-------------+-----------+

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.
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.

The decrypted contents of E are depicted below (whitespace has been added to improve readability). The presence of the attribute V indicates that the DTLS PSK Transfer is used to convey the session key (cf. Section 6.1).
6. DTLS PSK Generation Methods

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

6.1. DTLS PSK Transfer

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

The DTLS PSK transfer method is defined as follows:

- SAM generates the DTLS PSK using an algorithm of its choice.
- SAM MUST include a representation of the DTLS PSK in Face and encrypt it together with all other information in Face with a key K(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).
- SAM MUST include a representation of the DTLS PSK in the Verifier.
- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- S MUST decrypt Face using K(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 CAM, e.g. if SAM only wants 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 CAM 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
```


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 CAM contains an access ticket that is relayed back to CAM.
C --&gt; S
PUT a/switch2941 [Mid=1234]
Content-Format: application/senml+json
{"e": [{"bv": "1"}]}

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

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

CAM --&gt; 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 &lt;-- 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'7ba4d9e287c8b69dd52fd3498fb8d26d9503611917b014ee6ec2a570d857987a'
}

C &lt;-- 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'7ba4d9e287c8b69dd52fd3498fb8d26d9503611917b014ee6ec2a570d857987a'
}
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.
This example shows an Authorization request using implicit authorization. CAM initially requests the actions GET and POST on the resource "coaps://[2001:DB8::dcaf:1234]/a/switch2941". SAM returns a ticket that has no SAI field in its ticket Face, hence implicitly authorizing C.

CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor

{ SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 13]
}

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

{ F: {
  SAI: ["a/switch2941", 5],
  TS: 0("2013-07-04T21:33:11.930"),
  G: hmac_sha256
},
 V: h'c7b5774f2ddcd6548f4ad74b30a1b2e5
  b6b04e66a9995ed2545e5a06216c53d'
}

10.4. Implicit Authorization

11. Specific Usage Scenarios

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

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

11.1. Combined Authorization Manager and Client

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

```
CAM+C                  S                  SAM
|                    | <== DTLS chan. ==> |
| [Resource Req.--->] |                    |
| [ <-- 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.
11.2.1. Processing the Access Request Message

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

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

11.2.2. Creating the Ticket Transfer Message

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

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

11.3. Combined Server Authorization Manager and Server

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

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

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

13. IANA Considerations

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

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

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

Initial entries in this sub-registry are as follows:

+---------------+-------------+------------+
| 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.

<table>
<thead>
<tr>
<th>Media type</th>
<th>Encoding</th>
<th>Id.</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/dcaf+cbor</td>
<td>-</td>
<td>TBD1</td>
<td>[RFC-XXXX]</td>
</tr>
</tbody>
</table>

14. Acknowledgements

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

15.1. Normative References


15.2. Informative References

[I-D.bormann-core-ace-aif]  

[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]  

[I-D.ietf-ace-actors]  

[I-D.ietf-core-block]  

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

[I-D.ietf-cose-msg]  

[I-D.schmertmann-dice-codtls]  

[RFC5988]  
Appendix A. CDDL Specification

This appendix shows a formal specification of the DCAF messaging format using the CBOR data definition language (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
Abstract

The Delegated CoAP Authorization Framework (DCAF) defines how constrained devices can securely obtain security associations and authorization information from their respective less constrained devices, the Authorization Managers. In DCAF a constrained client requests an authorization ticket from the Server Authorization Manager (SAM) by contacting its own Client Authorization Manager (CAM). However, there may be cases where this approach is not applicable, e.g., because the client is not able to reach Authorization Managers in the Internet.

Specifically for these situations, this document defines the Server-Initiated Ticket Request (SITR) that specifies how a constrained server can request authorization tokens and securely obtain security associations and authorization information for mutual authenticated authorization with the client.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.
1. Introduction

See abstract.

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 RFC 2119 [RFC2119].
Readers are required to be familiar with the terms and concepts defined in [I-D.ietf-ace-actors] and [I-D.gerdes-ace-dcaf-authorize].

2. Protocol

2.1. Overview

The figure Figure 1 depicts the Sitr protocol flow:

```
CAM     C       CAM Info. ---->  S       SAM
        |                    | SI Acc. Req. -->
<=== (CAM/SAM Mutual Authentication and Authorization) ====>
<--------------- SIT Request ---->
SIT Grant ----------------------------------------------------->
<------ SIT Transf.  

<-- CAM Inf. Resp.
<= Authn Authz =>
Auth. Res. Req. ->
```

Figure 1: Protocol Overview

The authorization flow will then look as follows: C will send a CAM info message (maybe after first sending an unauthorized request to S that is denied) that contains its CAM address together with the request to this URI.

S will send a Server Initiated (SI) Access Request to SAM which includes the request and URI. SAM will contact CAM and determine if CAM has the respective permissions. The details of the communication between CAM and SAM are not in scope for this document, but CAM and SAM will mutually authenticate each other and then initiate a secure communication. Then SAM sends a SIT Request to CAM which contains the information from the Resource Request.

CAM checks if SAM is authorized according to COP’s authorization policies (mutual authorization). If this is the case, CAM creates a SI ticket. The ticket contains keying material for the secure communication between C and S and, if necessary, authorization.
information that reflect COP’s security policies for C. CAM sends this ticket to SAM, SAM includes server authorization information to the SI ticket if necessary and sends the ticket to S. S keeps one part of the ticket and sends the other part to C as a reply to C’s CAM Information message. With their respective part of the ticket, C and S can communicate securely.

2.2. CAM Information Message

C sends a CAM Information message to S to stimulate S to request a ticket for C. The message is constructed as follows:

1. The request method is FETCH (see [I-D.bormann-core-coap-fetch]).
2. The request URI is set to the URI of the requested resource.
3. The message payload contains a data structure that describes the action and resource for which C requests an access ticket as well as the CAM URI.

The data structure in the payload MUST contain:

1. The contact information for the CAM to use: a URI that specifies the CAM in charge of C.
2. A URI of the resource that C wants to access.
3. The actions that C wants to perform on the resource.

Figure 2 shows an example for a CAM Information message. (Refer to Section 3 for a detailed description of the available attributes and their semantics.)

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

Figure 2: CAM Info Message Example

Note: if used with object security, the FETCH request contains CBOR encoded message syntax structure (COSE), that conveys the application/dcaf+cbor payload.
The example shows a CAM information message payload for the resource "/s/tempC" on the Server "temp451.example.com". Requested operations in attribute SAI are GET and PUT.

The response to a CAM Information message is delivered by S back to C in a CAM Information Response message.

2.3. Server-Initiated Access Request Message

A server that receives a CAM Information message MAY use the information in the payload of the message to request a Server-Initiated (SI) Ticket for C. To do so, it contacts its own SAM. The SI 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 as well as the CAM URI.

The request URI identifies a resource at SAM for handling authorization requests from C. The URI SHOULD be announced by SAM in its resource directory as described in section 9 of [I-D.gerdes-ace-dcaf-authorize].

The message payload is constructed from the information that C has sent in its CAM Information message (see Section 2.2). The request MUST contain the attributes described in Section 2.2.

2.4. Server-Initiated Ticket Request Message

When SAM receives a Server-Initiated Access Request message from S and ROP specified authorization policies for S, SAM MUST check if the requested actions are allowed according to these policies. If all requested actions are forbidden, SAM 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, SAM either returns a cached response or attempts to create a SI Ticket Request message. The SI Ticket Request message MAY contain all actions requested by C since SAM will add SAI in the Ticket Transfer Message if ROP specified authorization policies (see Section 2.6).
SAM MAY return a cached response if it is known to be fresh according to Max-Age. SAM SHOULD NOT return a cached response if it expires in less than a minute.

If CAM does not send a cached response, it checks the content type of the request payload and validates that the payload contains at least the fields CAM and SAI. SAM MUST respond with 4.00 (Bad Request) if the type does not belong to the allowed content-types and if any of these fields is missing or does not conform to the format described in Section 3.

If the payload is correct, SAM creates a SIT Request message from the SI Access Request received from S as follows:

1. The destination of the Ticket Request message is derived from the "CAM" field that is specified in the Access Request message payload (for example, if the SI Access Request contained ‘CAM: "coaps://cam.example.com/authz"’, the destination of the Ticket Request message is cam.example.com).

2. The request method is POST.

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

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

CAM and SAM MUST be able to mutually authenticate each other, e.g. based on a public key infrastructure and MUST be able to communicate securely.

2.5. SI Ticket Grant Message

When CAM has received a SI Ticket Request message it has to evaluate the access request information contained therein. First, it checks whether the request payload is of a supported content type (see Section 4) and contains at least the fields CAM and SAI. CAM MUST respond with 4.00 (Bad Request) for CoAP (or 400 for HTTP) if any of these fields are missing or do not conform to the format described in Section 3.

CAM decides whether or not access is granted to the requested resource and then creates a SI Ticket Grant message that reflects the result. CAM initializes the access ticket comprised of a Face and the Client Information (CI).

The CI contains:
o the Client authorization information (CAI)

o a nonce

CI MAY additionally contain:

o a lifetime

o a CAI identifier (for revocation)

o keying material for C (if no key derivation method is used to generate the verifier in Face)

The Face at this point only comprises the verifier. CAM MAY generate the verifier using the method described in section 6.2 of [I-D.gerdes-ace-dcaf-authorize]. CAM MUST NOT include Server Authorization information (SAI) in the ticket Face.

The CI MUST be integrity-protected on the way to C. CAM MAY additionally protect the confidentiality of CI. If the CI contains keying material, CAM MUST ensure the confidentiality of CI.

The confidentiality of Face MUST be ensured on the way to SAM.

The SI Ticket Grant messages is then constructed as a success response (2.05 for CoAP, 200 for HTTP) with the ticket as content.

2.6. SI Ticket Transfer Message

The SI Ticket Transfer message is the response to the SI access request and delivers the ticket to S.

The CAI provided by CAM in the SI Ticket Grant message provide only the client-side permissions. If ROP defined access permissions for S, SAM MUST add server authorization information (SAI) to Face that reflect those policies. SAM MUST NOT include SAI that were provided by CAM.

SAM MUST provide for the confidentiality and integrity of Face when transmitting it to S. SAM MAY encrypt the CI.

2.7. CAM Information Response

When S receives a SI Ticket Transfer message, it MUST make sure that it contains the Face and the CI. If Face contains SAI, S MUST validate its authenticity and integrity. S keeps the ticket Face and sends the CI to C. S MAY transmit the answer to C’s initial request provided in the CAM Info message together with the CI.
When C receives the CAM Information Response, it MUST validate that the CI was generated by CAM and not modified. With the information in the CI, C can start a secure communication with S.

C MAY establish a security context with S using the verifier provided in the CI, e.g., by initiating a DTLS session with the verifier as the Pre-shared Key.

3. Payload Format

SITR uses the CBOR representation defined in DCAF (see section 5 of [I-D.gerdes-ace-dcaf-authorize]) and additionally defines a CBOR value for CAM:

```
+---------------+-----+
| Encoded Value | Key |
+---------------+-----+
| 13            | CAM |
+---------------+-----+
```

Table 1: SITR field identifiers encoded in CBOR

4. Content Types

The supported content types are:

- "application/dcaf+cbor"
- "application/cose+cbor"

5. IANA Considerations

None

6. Security Considerations

For solutions where the server requests the ticket for the client, most of the workload (send a message to the authorization manager, wait for the answer, keep state in the meantime) is on the server which makes it susceptible to DOS attacks. Therefore, as with all solutions state based on client requests, these solutions MUST NOT be used except in conjunction with appropriate mitigation. Where applicable, it is recommended to use DCAF instead, where the client has to request the ticket.
7. Acknowledgments

The author would like to thank Bert Greevenbosch, Olaf Bergmann and Carsten Bormann for their valuable input and feedback.

8. References

8.1. Normative References

[I-D.bormann-core-coap-fetch]
Bormann, C., "CoAP FETCH Method", draft-bormann-core-coap-fetch-00 (work in progress), October 2015.

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-03 (work in progress), September 2015.


8.2. Informative References

[I-D.ietf-ace-actors]

Author’s Address

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

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

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
Table of Contents

1. Introduction ........................................ 3
   1.1. Terminology .................................... 4
2. Architecture and High-level Problem Statement ........... 5
   2.1. Elements of an Architecture .................... 5
   2.2. Architecture Variants .......................... 8
   2.3. Problem statement ................................ 11
3. Security Objectives .................................. 12
   3.1. End-to-End Security Objectives in Multi-Hop Scenarios .... 13
4. Authentication and Authorization ........................ 13
5. Actors and their Tasks ................................ 15
   5.1. Constrained Level Actors ....................... 16
   5.2. Principal Level Actors .......................... 17
   5.3. Less-Constrained Level Actors .................. 17
6. Kinds of Protocols .................................. 18
   6.1. Constrained Level Protocols ..................... 18
   6.2. Less-Constrained Level Protocols ............... 19
7. Elements of a Solution ................................ 19
   7.1. Authorization ................................... 19
   7.2. Authentication .................................. 20
   7.3. Communication Security .......................... 21
   7.4. Cryptographic Keys ............................... 21
8. Assumptions and Requirements .......................... 22
   8.1. Architecture ..................................... 22
   8.2. Constrained Devices ............................. 23
   8.3. Authentication ................................... 24
   8.4. Server-side Authorization ........................ 24
   8.5. Client-side Authorization Information ........... 24
   8.6. Server-side Authorization Information ........... 25
   8.7. Resource Access .................................. 25
   8.8. Keys and Cipher Suites .......................... 26
   8.9. Network Considerations ........................... 26
   8.10. Legacy Considerations ............................ 26
9. Security Considerations ................................ 27
   9.1. Physical Attacks on Sensor and Actuator Networks .... 27
   9.2. Time Measurements ................................ 28
10. IANA Considerations .................................. 29
11. Acknowledgements .................................... 29
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.

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

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

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

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.

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

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

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

1.1. Terminology

Readers are 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]; the latter also defines additional terms such as "endpoint".

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

In addition, this document uses the following terminology:

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

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.


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


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

- An endpoint may host functionality of one or more actors.
- 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.

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

```
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Endpoint</td>
<td></td>
</tr>
</tbody>
</table>
```

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.
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 segregation of the control information provided by each principal.)
Figure 3 shows all three levels considered in this document. Note that the vertical arrows point down to illustrate exerting control and providing support; this is complemented by information flows that often are bidirectional. Note also that not all entities need to be ready to communicate at any point in time; for instance, RqP may have provided enough information to CAS that CAS can autonomously negotiate access to RS with AS for C based on this information.

2.2. Architecture Variants

The elements of the architecture described above are 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:
If RS is located on a more powerful device, it can be combined with AS:
If C and RS have the same principal, CAS and AS can be combined.

---

Figure 5: Combined AS and RS

---

Figure 6: CAS combined with AS
2.3. Problem statement

We now formulate the problem statement in terms of the information flows the architecture focuses on.

The interaction with the nodes on the principal level, RO and RqP, is not involving constrained nodes and therefore can employ an existing mechanism. The less-constrained nodes, CAS and AS, support the constrained nodes, C and RS, with control information, for example permissions of clients, conditions on resources, attributes of client and resource servers, keys and credentials. This control information may be rather different for C and RS, 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 potential information flows are shown in Figure 7. The direction of the vertical arrows expresses the exertion of control; actual information flow is bidirectional.

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

```
| CAS |                   | AS |
-----|-------------------|-----|
| a   | b = requests for control info | a   |
|  ^   | v                   |  ^  |
| C   | ------ request -------> | RS  |
|     | <------ response ------|     |
```

Figure 7: Information flows that need to be protected

- 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.

- Any necessary keys/credentials for protecting the interaction between the potentially constrained nodes will need to be established and maintained as part of a solution.

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 accountability and third-party verifiability. These additional objectives are not directly related to authorization and thus are not in scope of this draft, but may nevertheless be relevant. Accountability and third-party verifiability may require authentication on a device level, if it is necessary to determine which device performed an action. In other cases it may be more important to find out who is responsible for the device’s actions. See also Section 4 for more discussion about authentication and authorization.

The security objectives and their relative importance differ for the various constrained environment applications and use cases [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. For example, if RqP requires the integrity of representations of a resource R that RS is hosting, both C and RS need to partake in integrity-protecting the transmitted data. Moreover, RS needs to protect any write access to this resource as well as to relevant other resources (such as configuration information, firmware update resources) to prevent unauthorized users from manipulating R.
3.1. End-to-End Security Objectives in Multi-Hop Scenarios

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

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

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

- To enable a trustworthy publication service, a pub-sub broker may be untrusted with the plaintext content of a publication (confidentiality), but required to verify that the publication is performed by claimed publisher and is not a replay of an old publication (authenticity/integrity).

- To comply with requirements of transparency, a gateway may be allowed to read, verify (authenticity) but not modify (integrity) a resource representation which therefore also is end-to-end integrity protected from the server towards a client behind the gateway.

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

4. Authentication and Authorization

Server-side authorization solutions aim at protecting the access to items of interest, for instance hardware or software resources or data: They enable the resource owner to control who can access it and how.
To determine if an entity is authorized to access a resource, an authentication mechanism is needed. According to the Internet Security Glossary [RFC4949], authentication is "the process of verifying a claim that a system entity or system resource has a certain attribute value." Examples for attribute values are the ID of a device, the type of the device or the name of its owner.

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

A more detailed discussion of the tasks the actors have to perform in order to achieve specific security objectives is provided in [I-D.gerdes-ace-tasks].

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:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including access requests.
- Validate that an entity is an authorized server for R.

RS performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including responses to access requests.
- 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.)
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, such as with whom C is allowed to communicate. By definition, C and RqP belong to the same security domain.

RqP must fulfill the following task:

- 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:

- 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:

- Validate on the client side that an entity has certain attributes.
- Obtain authorization information about an entity from C’s principal (RqP) and provide it to C.
- Negotiate means for secure communication to communicate with C.

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

AS performs the following tasks:

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

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:

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

- 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.

- 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.

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

- 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:

- 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.

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

- 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, for example by encrypting the resource representation the RS only providing access to those who possess the key to decrypt.

- C may need to authenticate RS, in order to ensure that it is interacting with the right resources. Alternatively C may just verify the integrity of a received resource representation.

- 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:

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

- An alternative is object security at application layer, for instance 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.

- 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 such as in terms of deployment.

**Key Establishment**

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

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

8. Assumptions and Requirements

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

8.1. Architecture

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

- RS hosting resources, and responding to access requests
- C requesting access to resources
- AS supporting the access request/response procedure by providing authorization information to RS
  * AS may 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.
- 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.
- 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

- 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

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

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

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

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

- 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

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

- 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.
- The credentials presented by C may have been provided by CAS.
- The underlying authorization decision is taken either by AS or RS.
- 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.
- Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (for instance, resource descriptions).
- The solution may need to be able to support the delegation of access rights.

8.5. Client-side Authorization Information

- 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

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

8.7. Resource Access

- Resources are accessed in a RESTful manner using GET, PUT, POST, DELETE.
- 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.
- 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.
- RS needs to be able to verify that the request comes from an authorized client.
C needs to be able to verify that the response to a request comes from the intended RS.

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

8.8. Keys and Cipher Suites

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.

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

The access request/response can be protected with symmetric and/or asymmetric keys.

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

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

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

8.9. Network Considerations

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).

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

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

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

A solution may consider interworking with existing infrastructure.
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:

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

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

- Instead of eavesdropping the sensor data or attacking the authorization system to gain access to the data, the attacker could make its own measurements on the physical object.
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]

[I-D.garcia-core-security]

[I-D.gerdes-ace-tasks]

[I-D.hardjono-oauth-umacore]
[I-D.ietf-ace-usecases]

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

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

[OSCAR]

[RFC2904]

[RFC4120]

[RFC4949]

[RFC5246]

[RFC6347]

[RFC6749]


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
Farogatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Authorization for the Internet of Things using OAuth 2.0

draft-seitz-ace-oauth-authz-00

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of
Internet-Draft         OAuth 2.0 IoT Authorization          October 2015

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

Table of Contents

1.  Introduction .................................................. 3
2.  Terminology ................................................... 4
3.  Overview ....................................................... 4
   3.1. OAuth 2.0 ................................................. 5
   3.2. CoAP .................................................... 7
   3.3. Object Security .......................................... 8
4.  Protocol Interactions .......................................... 9
5.  OAuth 2.0 Profiling ........................................... 11
   5.2. Authorization Information Resource at the Resource Server 12
   5.3. Authorization Information Format .......................... 13
   5.4. CBOR Data Formats ....................................... 13
   5.5. CBOR Web Token .......................................... 13
6.  Deployment Scenarios ........................................... 13
   6.1. Client and Resource Server are Offline .................. 14
   6.2. Resource Server Offline .................................. 17
   6.3. Token Introspection with an Offline Client .............. 21
   6.4. Always-On Connectivity ................................... 25
   6.5. Token-less Authorization .................................. 25
   6.6. Securing Group Communication ............................. 28
7.  Security Considerations ........................................ 29
8.  IANA Considerations ............................................ 29
9.  Acknowledgments ................................................ 29
10. References ..................................................... 30
    10.1. Normative References ................................... 30
    10.2. Informative References .................................. 31
Appendix A. Design Justification ................................. 32
Appendix B. Optimizations ......................................... 34
Appendix C. CoAP and CBOR profiles for OAuth 2.0 ............... 35
   C.1. Profile for Token resource ............................... 35
      C.1.1. Token Request ....................................... 35
      C.1.2. Token Response ...................................... 37
   C.2. CoAP Profile for OAuth Introspection ..................... 38
      C.2.1. Introspection Request ............................... 38
      C.2.2. Introspection Response .............................. 39
Appendix D. CBOR Web Token (CWT) .................................. 41
   D.1. Claim Names ............................................... 41
      D.1.1. iss (Issuer) Claim ................................... 41
      D.1.2. sub (Subject) Claim ................................... 42
Authorization is the process of deciding what an entity ought to be allowed to do. Managing authorization information for a large number of devices and users is often a complex task where dedicated servers are used.

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

We envision that end consumers and enterprises will want to manage their Internet of Things (IoT) devices in the same style and this desire will increase with the number of devices that need to be managed and controlled. The IoT devices may be constrained in various ways including processing, memory, code, energy, etc., as defined in [RFC7228], and the different IoT deployments present a continuous range of device and network capabilities. Taking energy consumption as an example: At one end there are energy-harvesting or battery powered devices which have a tight power budget, on the other end there are mains-connected devices which are not constrained in terms of power, and all levels in between. Thus IoT devices are very different in terms of available processing and message exchange capabilities.

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

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

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

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

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

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

3. Overview

This specification describes a framework for authorization in the Internet of Things consisting of a set of building blocks.

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

Another building block is the lightweight web transfer protocol CoAP [RFC7252] for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP which further reduces overhead and message exchanges. When CoAP is used over UDP, transport layer security is provided by DTLS 1.2 [RFC6347] instead of TLS 1.2 [RFC5246].
A third building block is CBOR [RFC7049] for encodings where JSON [RFC7159] is not sufficiently compact. CBOR is a binary encoding designed for extremely small code size and fairly small message size. OAuth 2.0 allows access tokens to use different encodings and this document defines such an alternative encoding. The COSE message format [I-D.ietf-cose-msg] is also based on CBOR.

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

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

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

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

3.1. OAuth 2.0

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

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

Access Tokens:

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

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

Proof of Possession Tokens:

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

The proof-of-possession (PoP) security concept assumes that the AS
acts as a trusted third party that binds keys to access tokens.
These so called PoP keys are then used by the client to
demonstrate the possession of the secret to the RS when accessing
the resource. The RS, when receiving an access token, needs to
verify that the key used by the client matches the one included in
the access token. When this memo uses the term "access token" it
is assumed to be a PoP token unless specifically stated otherwise.

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

Symmetric PoP key:

The AS generates a random symmetric PoP key, encrypts it for
the RS and includes it inside an access token. The PoP key
is also encrypted for the client and sent together with the
access token to the client.

Asymmetric PoP key:

An asymmetric key pair is generated on the client and the
public key is sent to the AS (if it does not already have
knowledge of the client’s public key). Information about
the public key, which is the PoP key in this case, is then
included inside the access token and sent back to the
requesting client.

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

Scopes and Permissions:

In OAuth 2.0, the client specifies the type of permissions it is seeking to obtain (via the scope parameter) in the access request. In turn, the AS may use the "scope" response parameter to inform the client of the scope of the access token issued. This memo uses CBOR encoded messages defined in Appendix C to request scopes and to be informed what scopes the access token was actually authorized for by the AS.

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

Claims:

The information carried in the access token in the form of type-value pairs is called claims. An access token may for example include a claim about the AS that issued the token (the "iss" claim) and what audience the access token is intended for (the "aud" claim). The audience of an access token can be a specific resource or one or many resource servers. The resource owner policies influence the what claims are put into the access token by the authorization server.

While the structure and encoding of the access token varies throughout deployments, a standardized format has been defined with the JSON Web Token (JWT) [RFC7519] where claims are encoded as a JSON object. In Appendix D we define a CBOR version of JWT that we call CBOR Web Token (CWT).

Introspection:

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

3.2. CoAP
CoAP is an application layer protocol similar to HTTP, but specifically designed for constrained environments. CoAP typically uses datagram-oriented transport, such as UDP.

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

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

3.3. Object Security

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

Object Security of CoAP (OSCOAP)

OSCOAP [I-D.selander-ace-object-security] is a method for protecting CoAP request/response message exchanges, including CoAP payloads, CoAP header fields as well as CoAP options. OSCOAP provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages.

A CoAP message protected with OSCOAP contains the CoAP option "Object-Security" which signals that the CoAP message carries a COSE message ([I-D.ietf-cose-msg]). OSCOAP defines a profile of COSE which includes replay protection.

Object Security of Content (OSCON)

For the case of wrapping of application layer payload data ("content") only, such as resource representations or claims of access tokens, the same COSE profile can be applied to obtain end-to-end confidentiality, integrity and replay protection. [I-D.selander-ace-object-security] defines this functionality as Object Security of Content (OSCON).

In this case, the message is not bound to the underlying application layer protocol and can therefore be used with HTTP, CoAP, Bluetooth Smart, etc. Whereas OSCOAP integrity protects specific CoAP message meta-data like request/response code, and
binds a response to a specific request, since OSCON protects only payload/content, those security features are lost. The advantages are that an OSCON message can be passed across different protocols, from request to response, and used to secure group communications.

4. Protocol Interactions

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

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

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

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

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

At the start of the protocol there is an optional discovery step where the client discovers the resource server and the resources this server hosts. In this step the client might also determine what permissions are needed to access the protected resource. The exact procedure depends on the protocols being used and the specific deployment environment. In Bluetooth Smart, for example, advertisements are broadcasted by a peripheral, including information about the supported services. In CoAP, as a second example, a client can makes a request to "/.well-known/core" to obtain information about available resources, which are returned in a standardized format as described in [RFC6690].
Figure 1: Overview of the basic protocol flow

Requesting an Access Token (A):

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

Access Token Response (B):

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

Resource Request (C):

The client interacts with the RS to request access to the protected resource and provides the access token. The protocol to use between the client and the RS is not restricted to CoAP; HTTP, HTTP/2, Bluetooth Smart etc., are also possible candidates.
Depending on the device limitations and the selected protocol this exchange may be split up into two phases:

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

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

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

Token Introspection Request (D):

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

Note that token introspection is an optional step and can be omitted if the token is self-contained and the resource server is prepared to perform the token validation on its own.

Token Introspection Response (E):

The AS validates the token and returns the claims associated with it back to the RS. The RS then uses the received claims to process the request to either accept or to deny it.

Protected Resource (F):

If the request from the client is authorized, the RS fulfills the request and returns a response with the appropriate response code. The RS uses the dynamically established keys to protect the response, according to used communication security protocol.

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


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

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

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

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

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

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

5.2. Authorization Information Resource at the Resource Server

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

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

5.3. Authorization Information Format

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

5.4. CBOR Data Formats

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

5.5. CBOR Web Token

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

6. Deployment Scenarios

There is a large variety of IoT deployments, as is indicated in Appendix A, and this section highlights common variants. This section is not normative but illustrates how the framework can be applied.
For each of the deployment variants there are a number of possible security setups between clients, resource servers and authorization servers. The main focus in the following subsections is on how authorization of a client request for a resource hosted by a RS is performed. This requires us to also consider how these requests and responses between the clients and the resource servers are secured.

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

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

6.1. Client and Resource Server are Offline

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

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

This example shows the interactions between a client, the authorization server and a temperature sensor acting as a resource server. Message exchanges A and B are shown in Figure 2.

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

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

B: The AS responds with a PoP token and client information. The PoP token contains the public key of the client, while the client information contains the public key of the RS. For communication security this example uses DTLS with raw public keys between the client and the RS.
Note: In this example we assume that the client knows what resource it wants to access, and is therefore able to request specific audience and scope claims for the access token.

<table>
<thead>
<tr>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
</tr>
<tr>
<td>A: +--------&gt;</td>
</tr>
<tr>
<td>POST</td>
</tr>
<tr>
<td>Header: POST (Code=0.02)</td>
</tr>
<tr>
<td>Uri-Path: &quot;token&quot;</td>
</tr>
<tr>
<td>Payload: &lt;Request-Payload&gt;</td>
</tr>
<tr>
<td>B: &lt;--------+</td>
</tr>
<tr>
<td>2.05</td>
</tr>
<tr>
<td>Header: 2.05 Content</td>
</tr>
<tr>
<td>Content-Type: application/cbor</td>
</tr>
<tr>
<td>Payload: &lt;Response-Payload&gt;</td>
</tr>
</tbody>
</table>

Figure 2: Token Request and Response Using Client Credentials.

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

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

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

Figure 3: Request and Response Payload Details.

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

```
{
  "kid" : b64‘c29tZSBwdWJsaWMga2V5IGlk’,
  "kty" : "EC",
  "cry" : "P-256",
  "x" : b64‘MKBCTNIcKUSDii1lySs3526iDZ8AiTo7Tu6KPAqv7D4’,
```

"y" : b64'4Etl6SRW2YiLUrN5vfVHuhp7x8Px1tmWW1bbM4IFyM'
}

Figure 4: Public Key of the RS.

{
"aud" : "tempSensorInLivingRoom",
"iat" : "1360189224",
"cnf" : {
  "jwk" : {
    "kid" : b64'1Bg8vub9tLe1gHMzV76e8",
    "kty" : "EC",
    "crv" : "P-256",
    "x" : b64'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
    "y" : b64'x_FEzRu9m36HLN_tue659LNPw6pcyStikYjKAtIa0'
  }
}

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

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

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

The client and resource server run the DTLS handshake using the raw public keys established in step B and C.

The client sends the CoAP request GET to /temperature on RS over DTLS. The RS verifies that the request is authorized.

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: ------</td>
<td>POST</td>
<td>Header: POST (Code=0.02)</td>
</tr>
<tr>
<td></td>
<td>Uri-Path:&quot;authz-info&quot;</td>
<td>Payload: S1AV32hkKG ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(access token)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;--------+ Header: 2.04 Changed</td>
</tr>
</tbody>
</table>
6.2. Resource Server Offline

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

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

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

The following example shows interactions between a client (AC control unit), an offline resource server (temperature sensor) and an authorization server. The message exchanges A and B are shown in Figure 8.
A: The client sends the request POST to /token at AS. The request contains the Audience parameter set to "tempSensor109797", a value that the temperature sensor identifies itself with. The scope the client want's the AS to authorize the access token for is "owner", which means that the token can be used to both read temperature data and upgrade the firmware on the RS. The AS evaluates the request and authorizes the client to access the resource.

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

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

Figure 8: Token Request and Response

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

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

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

Figure 9: Request and Response Payload for RS offline

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

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

key:
{
  "alg": "AES_128_CCM_8",
  "kid": b64'U29tZSBLZXkgSWQ",
  "k": b64'ZoRSOrFzN_FzUA5XXMYoVHyzff5oRJxl-IXRtztJ6uE'
}

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

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

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

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

<table>
<thead>
<tr>
<th>Resource</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: +-------&gt; POST</td>
<td>Header: POST (Code=0.02)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uri-Path:&quot;authz-info&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payload: &lt;Access Token&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;-------</td>
<td>Header: 2.04 Changed</td>
<td></td>
</tr>
<tr>
<td>2.04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Access Token provisioning to RS

<table>
<thead>
<tr>
<th>Resource</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>+--------&gt; Header: GET (Code=0.01)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>Object-Security:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(&lt;seq&gt;,&lt;cid&gt;,[Uri-Path:&quot;tempC&quot;],&lt;tag&gt;)</td>
<td></td>
</tr>
<tr>
<td>F: &lt;------- Header: 2.05 Content</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.05</td>
<td>Object-Security:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(&lt;seq&gt;,&lt;cid&gt;,[22.7 C],&lt;tag&gt;)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 12: Resource request and response protected by OSCOAP

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
</tr>
<tr>
<td>A: +--------&gt;</td>
</tr>
<tr>
<td>POST</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B: &lt;---------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Token Request and Response using Client Credentials.
Authorization consent from the resource owner can be pre-configured, but it can also be provided via an interactive flow with the resource owner. An example of this for the key fob case could be that the resource owner has a connected car, he buys a generic key that he wants to use with the car. To authorize the key fob he connects it to his computer that then provides the UI for the device. After that OAuth 2.0 implicit flow is used to authorize the key for his car at the the car manufacturers AS.

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

Request-Payload:

```json
{
    "grant_type" : "token",
    "aud" : "lock0fDoor4711",
    "client_id" : "myclient",
}
```

Response-Payload:

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

Figure 14: Request and Response Payload for C offline

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

C: Next, the client POSTs the access token to the /authz-info resource in the RS. This is a plain CoAP request, i.e. no DTLS/OSCOAP between client and RS. Since the token is an opaque string, the RS cannot verify it on its own, and thus defers to respond the client with a status code until step E and only acknowledges on the CoAP message layer (indicated with a dashed line).

<table>
<thead>
<tr>
<th>Resource</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>C: +----------&gt;</td>
<td>POST</td>
<td>Header: POST (T=CON, Code=0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Token 0x2a12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Uri-Path:&quot;authz-info&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload: SlAV32hkKG ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(access token)</td>
</tr>
</tbody>
</table>
Figure 15: Access Token provisioning to RS

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

E: The AS provides the introspection response containing claims about the token. This includes the confirmation key (cnf) claim that allows the RS to verify the client’s proof of possession in step F.

After receiving message E, the RS responds to the client’s POST in step C with Code 2.04 (Changed), using CoAP Token 0x2a12. This step is not shown in the figures.

Figure 16: Token Introspection for C offline

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

Request-Payload:
{
    "token" : b64’S1AV32hkKG...’,
    "client_id" : "myRS",
    "client_secret" : "ytrewq"
}

Response-Payload:
{


"active" : true,
"aud" : "lockOfDoor4711",
"scope" : "open, close",
"iat" : 1311280970,
"cnf" : {
  "ck" : b64'JDLUhTMjU2IiwiY3R5Ijoi ...'
}

Figure 17: Request and Response Payload for Introspection

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

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

Resource

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td>PUT</td>
</tr>
<tr>
<td>Header: PUT (Code=0.03)</td>
<td>Header: PUT (Code=0.03)</td>
</tr>
<tr>
<td>Object-Security:</td>
<td>Object-Security:</td>
</tr>
<tr>
<td>(&lt;seq&gt;,&lt;cid&gt;,[Uri-Path:&quot;lock&quot;, 1],&lt;tag&gt;)</td>
<td>(&lt;seq&gt;,&lt;cid&gt;,[Uri-Path:&quot;lock&quot;, 1],&lt;tag&gt;)</td>
</tr>
</tbody>
</table>

Figure 18: Resource request and response protected by OSCOAP

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

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

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

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

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

6.5. Token-less Authorization

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

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

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

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

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

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

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

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

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

Figure 19: Token Request and Response

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

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

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

Figure 20: Request and Response Payload for RS severely constrained

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

key :
{
  "alg" : "AES_128_CTR_ECDSA",
  "kid" : b64'c29tZSBvdGlhOiBSZXKgaWQ';
  "k" : b64'ZoRSOrFzN_FzUA5XKMYoVHzzff5oRJx1-IXRtztJ6uE',
  "crv" : "P-256",
  "x" : b64'MKBCTNiKUSDli1lySs3526iZ28AiTo7Tu6KPAqv7D4',
  "y" : b64'4Et16SRW2YiLUrN5zfVhup7x8PxtmWlbbM4IFyM'
}

Figure 21: The 'key' Parameter

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

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

Message exchanges C and F are shown in Figure 22.

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

  F: The MB responds with the cached object secured content.
### Figure 22: Sensor measurement protected by COSE

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

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

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

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

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

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

7. Security Considerations

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

8. IANA Considerations

TBD

9. Acknowledgments

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

10.1. Normative References

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

[I-D.ietf-cose-msg]

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

[I-D.ietf-oauth-pop-architecture]

[I-D.ietf-oauth-pop-key-distribution]

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

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


10.2. Informative References


Appendix A.  Design Justification

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

Common IoT constraints are:

Low Power Radio:

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

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

Small Amount of Memory:

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

User Interface Limitations:

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

Communication Constraints:

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

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

Appendix B. Optimizations

This section sketches some potential optimizations to the presented solution.

Access token in DTLS handshake

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

Reference token and introspection

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

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

Reference token in CoAP option

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

One way to achieve this is to define a new CoAP option for carrying reference tokens, called "Ref-Token" as shown in the example in Figure 23.
Appendix C. CoAP and CBOR profiles for OAuth 2.0

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

C.1. Profile for Token resource

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

C.1.1. Token Request

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

grant_type:

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

client_id:

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

   OPTIONAL. The client secret.

scope:

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

aud:

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

alg:

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

key:

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

The parameters defined above use the following CBOR major types.

```
<table>
<thead>
<tr>
<th>Value</th>
<th>Major Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>grant_type</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>client_id</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>client_secret</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>scope</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>aud</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>alg</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>key</td>
</tr>
</tbody>
</table>
```

Figure 24: CBOR mappings used in token requests
C.1.2. Token Response

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

access_token:
  REQUIRED. The access token issued by the authorization server.

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

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

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

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

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

The parameters defined above use the following CBOR major types.

<table>
<thead>
<tr>
<th>Value</th>
<th>Major Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>access_token</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>token_type</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>key</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>csp</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>scope</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>alg</td>
</tr>
</tbody>
</table>
C.2. CoAP Profile for OAuth Introspection

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

C.2.1. Introspection Request

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

- **token**: REQUIRED. The string value of the token.
- **resource_id**: OPTIONAL. A service-specific string identifying the resource that the client doing the introspection is asking about.
- **client_id**: OPTIONAL. The client identifier issued to the holder of the token (client or RS) during the registration process.
- **client_secret**: OPTIONAL. The client secret.

The parameters defined above use the following CBOR major types:

<table>
<thead>
<tr>
<th>Value</th>
<th>Major Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>token</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>resource_id</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>client_id</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>client_secret</td>
</tr>
</tbody>
</table>

Figure 25: CBOR mappings used in token responses
C.2.2. Introspection Response

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

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

Introspection responses in CBOR use the following mappings:

active:

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

scope:

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

client_id:

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

username:

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

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

exp:

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

iat:

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

nbf:

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

sub:

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

aud:

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

iss:

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

cti:

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

The parameters defined above use the following CBOR major types:

```
<table>
<thead>
<tr>
<th>Value</th>
<th>Major Type</th>
<th>Key</th>
</tr>
</thead>
</table>
```

Seitz, et al. Expires April 21, 2016 [Page 40]
CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. CWT is a profile of JSON Web Tokens that is optimized for constrained devices. The claims in a CWT are encoded in CBOR and COSE is used for signature and encryption. A claim is a piece of information asserted about a subject. A claim is represented as a name/value pair consisting of a Claim Name and a Claim Value.

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

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

D.1. Claim Names

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

D.1.1. iss (Issuer) Claim
The "iss" (issuer) claim identifies the principal that issued the CWT. The processing of this claim is generally application specific. The "iss" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

D.1.2. sub (Subject) Claim

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

D.1.3. aud (Audience) Claim

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

D.1.4. exp (Expiration Time) Claim

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

D.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim identifies the time before which the CWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.
D.1.6. iat (Issued At) Claim

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

D.1.7. cti (CWT ID) Claim

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

D.1.8. cnf (Confirmation) Claim

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

D.1.9. cks (COSE Key Structure) Claim

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

D.1.10. aif (Authorization Information Format) Claim

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

D.2. CBOR major types for Claims

<table>
<thead>
<tr>
<th>Value</th>
<th>Major Type</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>iss</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>sub</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>aud</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>nonce</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>exp</td>
</tr>
</tbody>
</table>
Figure 28: CBOR Mappings used in CWT Access Tokens.

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

D.3. CBOR Web Token Example

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

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

Figure 29: CWT Example in the CBOR Diagnostic Notation.

Authors' Addresses

Ludwig Seitz
SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig@sics.se