

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

O. Bergmann
S. Gerdes
Universitaet Bremen TZI
October 19, 2015

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.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Overview	4
2.1. Sending Authorized Requests	4
2.2. Responding to an Authorized Request	5
3. Establishing a Security Context	6
3.1. Unauthorized Resource Request Message	6
3.2. SAM Information Message	7
3.3. Access Request	8
3.4. Ticket Request Message	10
3.5. Ticket Grant Message	10
3.6. Ticket Transfer Message	11
3.7. Security Association between C and S	12
4. Piggybacked Protected Content	13
5. CoAP Options Authorization and Authorization-Format	14
6. Canonicalization of the CoAP Message Header	14
7. The "auth-info" Link Relation	16
8. Security Considerations	17
9. IANA Considerations	17
9.1. CoAP Option Registration	17
10. Acknowledgements	17
11. References	17
11.1. Normative References	17
11.2. Informative References	18
Authors' Addresses	19

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:

- o Authenticated exchange of authorization information.
- o Simplified authentication on constrained nodes by handing the more sophisticated authentication work over to less-constrained devices.
- o Support of secure constrained device to constrained device communication.
- o Authorization policies of the principals of both participating parties are ensured.
- o Simplified authorization mechanism for cases where implicit authorization is sufficient.
- o Can be made to work just using symmetric encryption on the constrained nodes.
- o 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. [_1] 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.

```
PUT /r
Content-Format: application/cose+cbor
[ h'a10300',                # protected { content_type: text/plain }
  { alg: HMAC 256/256,      # unprotected
    kid: h'3233386473613239' # context identifier: "238dsa29"
  },
  h'48656c6c6f20576f726c6421', # 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
Authorization: [ h'',          # protected (empty)
  { alg: HMAC 256/256,        # unprotected
    kid: h'3233386473613239'  # context identifier: "238dsa29"
  },
  nil,                        # payload (empty)
  h'....',                    # tag: HMAC(options+protected, secret)
  [ [ h'', {}, h'' ] ]       # recipients
]
```

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'77cd8a8047b7af7113bb074bcc', # 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:

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

- o S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access 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.)

4.01 Unauthorized

Content-Format: application/dcaf+cbor

{SAM: "coaps://sam.example.com/authorize", TS: 168537,

A: [ct_cose_msg] }

Figure 4: SAM Information Payload Example

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

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
    616d706c652e636f6d2f6175746866f72
    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.)

```
POST /client-authorize
Content-Format: application/cose+cbor
[ h'a1031862',          # protected { content_type: application/dcaf+cbor }
  { alg: AES-CCM-16-64-128      # unprotected
    nonce: h'd6150b90e6f0eb5be42164062c', # nonce
  },
  h'TBD:encrypted payload w/ tag', # encrypted DCAF payload
  # recipients:
  [ [ h'',              # protected (absent for AE algorithm)
    { alg: A128KW,      # unprotected
      kid: h'383261622e6161733432' # context identifier: "82ab.aas42"
    },
    h'52ff9ed52d...'      # encrypted session key
  ] ]
]
```

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'a1031862',          # 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
    6d0f0565160e3ff7d3907441cdf44cc9'
  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:

- o SAM and S both generate the verifier using the information included in Face. They use an HMAC algorithm on Face with a shared key $K(\text{SAM}, S)$. The result serves as the content encryption key. How SAM and S exchange $K(\text{SAM}, S)$ is not in the scope of this document. They MAY use their preshared key as $K(\text{SAM}, S)$.
- o SAM MUST include a representation of the session key in the Verifier.
- o As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using protected channels.
- o SAM MUST NOT include a representation of the Verifier in Face.
- o 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
[ h'a1031862',      # protected { content_type: application/dcaf+cbor }
  { alg: HMAC 256/256 },      # unprotected
  h'{ SAI: [ "/s/tempC" ... ] }', # DCAF payload wrapped in CBOR binary
  h'.....',          # tag: HMAC(options+protected+payload, secret)
  [ [ h'', {}], h'' ] ]      # recipients
]

S --> C
2.01 Created
Content-Format: application/cose+cbor
Location-Path: 238dsa29
Authorization: [ h'a1031862',      # protected
  { alg: HMAC 256/256 },          # unprotected
  h'',                            # empty payload
  h'.....',                      # tag: HMAC(options+protected, secret)
  [ [ h'', {}], h'' ] ]          # recipients
]
```

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.

No.	C	U	N	R	Name	Format	Length	Default
64					Authorization	opaque	1-1034	(none)
65	x				Authorization-Format	uint	0-2	application/cose+cbor

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:

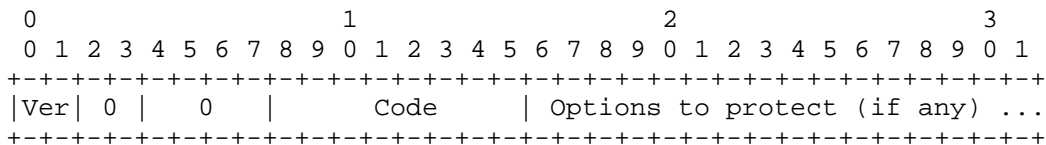


Figure 11: Canonicalized CoAP Message Header

As shown in Figure 11, only the version bits and the message code from the CoAP base header are relevant for integrity protection. [_3] From the list of options that a message might have, only the following options are to be included with the canonicalized message. [_4]

- o If-Match
- o Uri-Host
- o 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 Size1

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'a203766170706c69636174696f6e2f73656e6d6...',
  { alg: HMAC 256/256, # unprotected
    kid: h'3233386473613239' # context identifier: "238dsa29"
  },
  h'a2231a4eaecc5d....', # 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
;title="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"
;title="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.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

11.2. Informative References

- [I-D.ietf-ace-actors] Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.
- [I-D.ietf-cose-msg] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 2015.
- [I-D.jennings-core-senml] Jennings, C., Shelby, Z., Arkko, J., and A. Keranen, "Media Types for Sensor Markup Language (SENML)", draft-jennings-core-senml-02 (work in progress), October 2015.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<http://www.rfc-editor.org/info/rfc6838>>.

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

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 6, 2018

J. Cuellar
P. Kasinathan
Siemens AG
D. Calvo
Atos Research and Innovation
January 2, 2018

Privacy-Enhanced-Tokens (PAT) profile for ACE
draft-cuellar-ace-pat-priv-enhanced-authz-tokens-06

Abstract

This specification defines PAT, "Privacy-Enhanced-Authorization-Tokens", an efficient protocol and an unlinkable-token construction procedure for client authorization in a constrained environment. This memo also specifies a profile for ACE framework for Authentication and Authorization. The PAT draft uses symmetric cryptography, proof-of-possession (PoP) for a key owned by the client that is bound to an OAuth 2.0 access-token.

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 July 6, 2018.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. PAT Overview and Features	4
4. PAT Protocol	6
4.1. RS<->AS: Security-association-Setup	7
4.2. [C->RS : Resource-Request]	7
4.3. [RS->C : Un-Authorized-Request(AS-Info)]	7
4.4. C<->AS : Security-Association-Setup	9
4.5. C->AS : Access-Request	9
4.6. C<-AS : Access-Response	11
4.6.1. Access-Token construction:	12
4.6.2. Verifier or PoP key construction:	13
4.7. C->RS : Resource-Request	14
4.8. RS->C : Resource-Response	17
4.8.1. RS Response-codes to C RES-REQ:	19
4.9. Construction of Derived-Tokens (DT)	19
4.9.1. C->RS: Resource-Request via DT	19
4.9.2. RS->C : Resource-Response to DT	21
5. Security Considerations	21
5.1. Privacy Considerations	22
6. IANA Considerations	22
7. References	22
7.1. Normative References	22
7.2. Informative References	23
8. Acknowledgement	23
8.1. Copyright Statement	23
Appendix A. ACE profile Registration	23
Authors' Addresses	24

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.ietf-ace-actors], [I-D.ietf-ace-oauth-authz] and [KoMa2014]. This memo describes a way of constructing Tokens from an initial secret that can be used by clients and resource servers (or in some cases, more generally by arbitrary nodes) to delegate client authentication and authorization in a constrained environment to trusted and unconstrained authorization servers.

This draft uses the architecture of [draft-ietf-ace-actors] and [I-D.ietf-ace-oauth-authz], designed to help constrained nodes in authorization-related tasks via less-constrained nodes. Terminology for constrained nodes is described in [RFC7228]. A device (Client) that wants to access a protected resource on a constrained node (Resource Server) first has to gain permission in the form of a token from the Authorization Server. This memo also specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz].

The main goal of the PAT is to present methods for constructing authorization tokens efficiently with privacy features such as unlinkability. The CoAP protocol [RFC7252] MAY be used as the application layer protocol. The draft uses symmetric Proof-of-Possession keys [I-D.ietf-oauth-pop-architecture], CBOR web tokens (CWT) [draft-ietf-ace-cbor-web-token-05] claims to represent security claims together with CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] and Concise Binary Object Representation (CBOR) [RFC 7049].

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].

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 [RFC2119] significance.

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), resource owner (RO), resources (R) and the authorization server (AS).

- o Access-Token (AT): the access token is a token prepared by the AS for C. It represents the privileges granted by the RO to the C to perform actions over the Resources (R) on an RS.
- o Token (Tk): this token is prepared by the C, presented to the RS to access the resources (R) on RS. The Tk contains all information needed by the RS to verify that it was granted by AS. The Client derives Tk from the AT.

In version-5 of PAT draft the token names -- AT and Tk -- and their purposes were harmonized with [I-D.ietf-ace-oauth-authz].

3. PAT Overview and Features

The PAT protocol is designed to work with ACE framework [I-D.ietf-ace-oauth-authz] and ACE actors [I-D.ietf-ace-actors]. In this specification we assume the following:

- o A Resource Server (RS) has one or more resources (R) and it is registered with an Authorization Server (AS)
 - o The Authorization Server (AS) provides access-tokens for the clients to access resources of RS. The corresponding Resource Owner (RO) of the RS MAY assign allowed-permissions for the Clients in the AS.
 - o The RS is offline after commissioning, i.e., RS cannot make any introspective queries to the AS to verify the authorization information provided by the C.
 - o A Client (C) is either registered with an AS or it knows how to reach the RS for accessing the required resources.
- * To access a resource on a Resource Server (RS), a Client (C) should request an access-token (AT) from AS, either directly or using its Client Authorization Server (CAS). For the sake of simplicity, this memo does not include the actor CAS.

Based on the above assumptions, a simple PAT message flow can be described as follows: a C may perform a resource-request to RS without a valid access-token, the RS will reject and it may provide AS information to the C in the response. The C performs an Access-Request to AS to ask for an AT that allows accessing the required resource (R) on RS. The AS checks if C is allowed to access the resource (R) on RS or not, based on permissions assigned by the RO. If C has sufficient permissions, then AS generates an Access-Token (AT) plus proof-of-possession (PoP) key bounded to the access-token and a common secret (K) between AS and RS. AS sends both the AT and the PoP key to C via a secure channel. How this secure channel is created between AS and C is out of the scope of this draft. After receiving AT and PoP key, C performs a resource-request to RS by constructing token (Tk) from AT or by deriving Token. The RS can construct its own version of the PoP key from the AT and verifies the received AT. If it is valid, RS encrypts the response with the PoP key. At the end of this phase, both C and RS has established a common derived secret, the PoP key. Later, C can generate unlinkable tokens (Tk) from the initial AT as described in Section 4.9.

In particular, PAT is designed to be used in contexts where unlinkability (privacy) and efficiency are the main goals: the tokens

(Tk) convey only the assurance of the authorization claims of the clients. In particular, the procedure described in Section 4.9 enables the Tokens (Tk) to be constructed in such a way that they do not leak information about the correspondence of messages to the same Client or from the same access-token (AT). For example, if an eavesdropper observes the messages from different Clients to and from the Resource Servers, 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 from lower-layer protocols may leak some information, and this can be treated separately via other methods.

The main features of PAT protocol are described below:

- o The PAT method allows a RO, or an Authorization Server (AS) on its behalf, to authorize one or several clients (C) to access resources (R) on a constrained Resource Server (RS). The C can also be constrained devices. The Access-Token (AT) response from AS to C MUST be performed via secure channels.
- o The RO is able to decide (if he wishes: in a fine-grained way) which client under which circumstances may access the resources exposed by the RS. This can be used to provide consent (in terms of privacy) from RO.
- o The Access-Tokens (AT) are crafted in such a way that the client can derive Tokens (Tk) that allow demonstrating to RS its authorization claims. The message exchange between C and RS for the presentation of the tokens MAY be performed via insecure channels to enforce efficiency. But the payload content -- if the Client is performing a POST/PUT/DELETE request -- from C to RS or the response payload from RS to C MUST be encrypted.
- o The RS can derive the PoP key from the AT, which is received attached to the Resource Request message, and it encrypts the response using it.
- o The tokens (Tk) do not provide any information about any associated identities such as identifiers of the clients, of access-tokens (AT) and of the resource-server.
- o The tokens (Tk) are supported by a "proof-of-possession" (PoP) key and the initial access-token (AT). The PoP key allows an authorized entity (a client) to prove to the verifier (here, the RS), that C is indeed the intended authorized owner of the token and not simply the bearer of the token.

To be coherent with ACE Authorization framework [I-D.ietf-ace-oauth-authz], this draft also specifies an ACE profile to use PAT and for efficient encoding it uses CWT and COSE. The PAT profile is signaled when the C requests token from the AS or via RS in response to unauthorized request response. The PAT profile will cover all the requirements described in [I-D.ietf-ace-oauth-authz].

4. PAT Protocol

The detailed description of PAT protocol is presented in this Section 4. The PAT protocol includes three actors: the RS, the C, and the AS. PAT message flow is shown in Figure 1. Messages in [square brackets] mean they are optional.

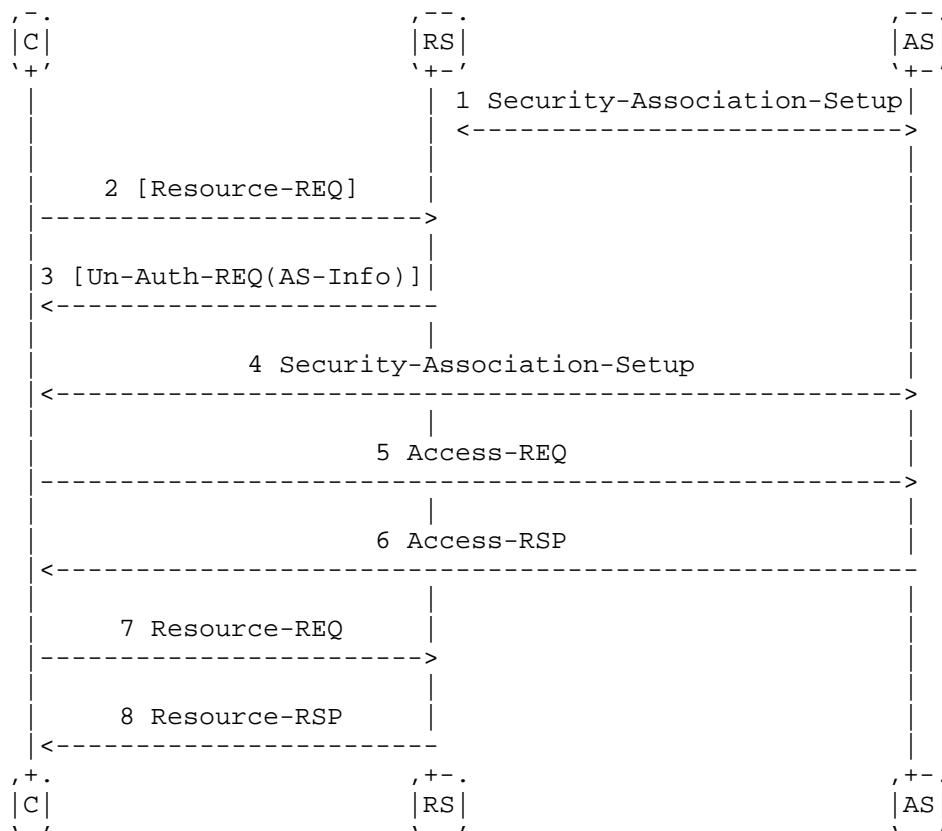


Figure 1: PAT protocol message flow

The following subsections describe the message flow in more detail, especially how the messages and tokens with PoP are constructed.

A PAT message sent from actor A to actor B is represented using the following notation: "A -> B : Message Name"

4.1. RS<->AS: Security-association-Setup

This memo assumes that the Resource Server (RS) and its Authentication Server (AS) share a long term shared secret (K), i.e., a shared key which MAY be implemented via USB (out of band methods) when device commissioning -- out of scope --. The shared secret (K) is used both by the AS and the RS to create proof-of-possession keys (PoP keys or verifiers).

We can also assume that the CAS and AS share a secure connection if CAS exist as an actor, e.g., DTLS. During the commissioning phase, RS registers the cryptographic algorithms and the parameters it supports. The internal clock of RS can be synchronized with the AS during the commissioning phase. Also, PAT supports the use of Lightweight Authenticated Time (LATE) Synchronization Protocol [I.D-draft-navas-ace-secure-time-synchronization].

4.2. [C->RS : Resource-Request]

Initially, a C may not have a valid access-token (AT) to access a protected resource (R) hosted in RS. The C might not also know the corresponding AS-information to request AT from AS. In this scenario, C may send a Resource-Request message to RS without a valid Token (Tk).

To enable resource discovery, RS may expose the URI `"/.well-known/core"` as described in [RFC6690], but this resource itself MAY be protected. Thus, C can optionally make a CoAP GET request to the URI `"/.well-known/core"`.

4.3. [RS->C : Un-Authorized-Request(AS-Info)]

Once RS receives a resource request from a C, it checks:

- o If C has attached a valid token (Tk) or not to the request. If C does not have a valid token (Tk), then RS MUST respond to C with 4.01 (Unauthorized request). Optionally, RS may include information about AS(AS-Info) which includes additional parameters (AS address) to reach the /token endpoint exposed by the AS. Note: this message is sent to any unauthorized Client, therefore it is recommended to include as less information as possible to identify AS.
- o If C has a valid access token, but not for the requested resource then RS MUST respond with 4.03 (Forbidden)

- o If C has a valid access token, but not for the method requested then RS MUST respond with 4.05 (Method Not Allowed)
- o If C has a valid access token, then RS must follow the procedure described in Section 4.8 to create a valid response to C.

Figure 2 shows the sequence of messages with detailed CoAP types between C and RS for the above Un-Authorized-Request:

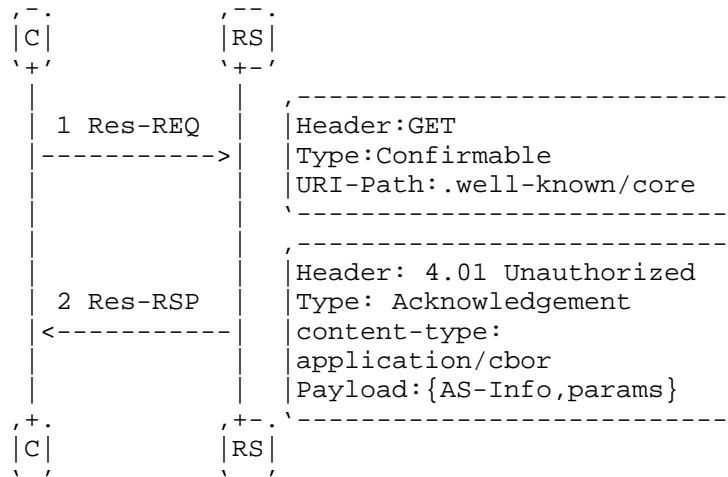


Figure 2: C<->RS Resource-Request and Unauthorized as response

The RS MAY send an Unauthorized response with additional information such as AS-Info and parameters (params). To mitigate attacks based on time synchronization, the Lightweight Authenticated Time (LATE) synchronization protocol [I.D-draft-navas-ace-secure-time-synchronization] MAY be used. In section 6.2 of [I.D-draft-navas-ace-secure-time-synchronization] Possible Scenarios, the scenario 1.2 of suits PAT protocol, an example of it is shown in figure 3.

The response payload MAY include AS information (AS-info) and LATE time synchronization's TIC information object such as key-reference ID (kid) shared secret between RS and AS, a nonce to prevent replay attacks and the message authentication codes (MAC) algorithm [optional] used for producing the MAC. It is recommended for RS to create a MAC tag for TIC parameters.

Figure 3 shows RS example response message to C encoded using CBOR [RFC7049] with pat-type="UnAuthReq".

```
Header: 4.01 (Unauthorized)
Content-Type: application/cbor+pat;
              pat-type="UnAuthReq"
Payload:
{
  AS-Info: "coaps://as.example.com/token",
  #protected
  TIC params:
  {
    nonce: 'rs-nonce..',
    kid: '.. ',
    [alg]: '.. ',
    TAG: '.. '
  }
}
```

Figure 3: AS information + LAtE time synchronization payload

4.4. C->AS : Security-Association-Setup

Before sending an access-request message, C must establish a secure channel with the AS. The C may be registered with the AS, as described in [I-D.ietf.ace-oauth-authz] or the C MAY have received AS-Info from RS as the answer to a previous Un-Authorized-Request.

The AS may have an access-control list defined by the RO for the authorized clients. With this access-control list, AS can verify if the client is allowed to establish a secure connection or not. If RO granted enough privileges to the client to access the requested resource (R) in RS, then AS establishes a confidential channel with C. How this secure connection (e.g., a DTLS channel) should be established is out of the scope of this memo.

Notice that, it is important to ensure that the connection between AS and C MUST be reliable and secure since the PAT protocol relies on the fact that the messages exchanged between C and AS are protected and confidential. If the Client is also a constrained device, then C may use DTLS-profile as described in [I.D-draft-gerdes-ace-dtls-authorize] to create the secure channel with the AS.

4.5. C->AS : Access-Request

Once C establishes a secure communication channel with AS, C sends an access-request (ACC-REQ) message to AS to the endpoint /token requesting an access token for RS as described in [I-D.ietf.ace-oauth-authz].

Optionally, the C includes as part of the Access-Request Message the details about the resources (R) or their corresponding URI with the operations it needs to access or perform on RS. If not AS should prepare an access token with default permissions. Fine grained access to resources (R) of RS depends on the infrastructure or services the RS offers. For example, if RS exposes different resources such as temperature and humidity, a generic access token may be granted by AS to C to access both resources on RS. On the other hand, the application developer or administrator may decide the access-rights based on application requirements.

Figure 4 shows an access-request message sent from C to AS to get an access token. The "aud" represents a specific resource R ("tempSensor") and "scope" represents the allowed actions that C aims to perform as described in [I-D.ietf-ace-oauth-authz] using CWT [I-D.ietf-ace-cbor-web-token]. The Scope parameter can be designed based on application requirements i.e., it can be "read" or "write" or methods such as "GET|POST" etc. If RS has included TIC information for time synchronization, then the C MUST include TIC object, including the MAC -- if included -- without any changes in the payload for access request.

How the client authenticates itself against the AS is out of the scope of this draft. Nevertheless, in the following example, the client presents the Client_Credentials i.e., password based authentication by presenting its client_secret (see section 2.3.1. of [RFC6749]).

```
Header: POST (Code=0.02)
Uri-Host: "coaps://as.example.com"
Uri-Path: "token"
Content-Type: "application/cbor+cwt+late ;
late-type=tic"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id": "client123",
  "client_secret": "Secret123",
  "aud" : "tempSensor",
  "scope": "GET|POST",
  ... omitted for brevity ...
  TIC params:
  {.. [if exist] ..
  nonce:'rs-nonce..', # same rs-nonce sent by RS
  kid: '..'
  }
  TAG: .. # TIC MAC tag produced by RS
        using the shared key kwith AS.
}
```

Figure 4: Example Client Access-Request message to AS

4.6. C<-AS : Access-Response

When AS receives an access-request message from a C, AS validates it and performs the following actions:

- o If the access request from C is valid (i.e., operations are covered by the privileges defined by the RO), then AS prepares the Access-Token (AT) and sends it with COAP response code 2.01 (Created).
- o If the Access-Request from C contains LAtE time synchronization TIC information object, then an appropriate response with TOC information object is included in the response as described in [I.D-draft-navas-ace-secure-time-synchronization].
- o If the client request is invalid then AS MUST send appropriate COAP error response code as specified in [I-D.ietf-ace-oauth-authz].

The Figure 5 shows the Access-Request from C to AS and the Access-Response from AS to C.

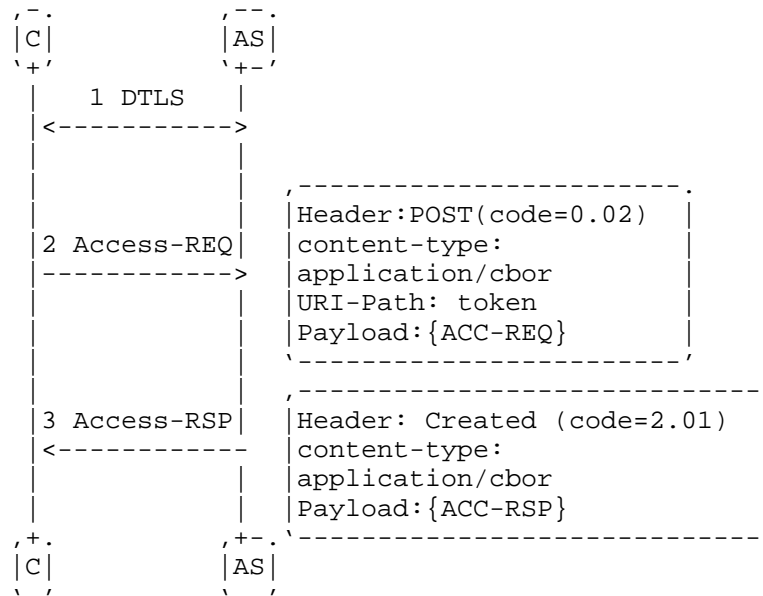


Figure 5: Access-Request and Access-Response

The AS constructs the Access-Token (AT) and the verifier (the symmetric PoP key) as the answer for a valid access request from C. The contents of the access-response (ACC-RSP) payload are logically split into two parts: the Access-Token (AT) and the Verifier (Symmetric PoP key).

4.6.1. Access-Token construction:

- o The Access-Token is constructed by AS using the CWT claim parameters. It represents the permissions granted to the Client.
 - * "iss" (issuer): AS identity
 - * "aud" (audience): resource server URI or specific resource URI for a fine-grained procedure.
 - * "exp" (Expiration Time): token expiration time
 - * "iat" (Issued At): token issued at time by AS
 - * "cti" CWT ID should be unique for every Access-Token.
 - * "scp" (Scope): Note that scp is not a CWT claim. It can specify allowed methods such as GET, POST, PUT or DELETE.

Other CWT claims are optional. It is recommended to avoid the CWT claim "sub" (subject) as it exposes client identity.

4.6.2. Verifier or PoP key construction:

- o Verifier (Symmetric PoP key): $G(K, \text{Access-Token})$. The Client will use the Verifier as the key material to communicate with the RS, i.e., if C wants to encrypt its payload, it uses the verifier as the key.
- * G : the MAC algorithm which is used to create the verifier, we propose Poly1305 [RFC7539]. Notice that G is a function which takes two parameters (key, data) as input and it produces a keyed digest as the output
- * K : the shared key between AS and RS
- * Access-Token: constructed using CWT claims as explained before

It is of special importance that the Access-Response message with the access token and the verifier MUST be sent to C through a secure channel -- in our example we considered a DTLS channel between C and AS --.

The time-synchronization between AS and RS MAY be implemented based on the application requirements using [I.D-draft-navas-ace-secure-time-synchronization].

The AS should specify required parameters as described in [I-D.ietf-ace-oauth-authz] such as the type of token, etc. Also, if the Access-Request from C does not include any profile, AS MUST signal the C to use appropriate or default profile that is supported by RS.

If the access-request message includes LATe TIC information, then AS MUST prepare TOC information and included it in the response. A MAC tag for TOC is created and appended in the response to prevent the client from tampering TOC information.

Figure 6 shows the example of an Access-Response sent from AS to C after successful validation of C's credentials which were presented using an Access-Request message.


```

Header: 2.01 (Created)
Content-Type: application/cbor+cwt+pat; pat-type="tk"
Location-Path: token/...
Payload:
{
  "access token": b64'SlAV32hkKG ...
  {
    "iss": https://as.example.com
    "aud": "tempSensor",
    "scp": "read",
    "iat": 1...,
    "cti": "..", # Unique can be a Sequence Number
    "exp": 5...,
    "alg": "chacha20/poly1305",
    "profile": "ace_pat"
  }
  "cnf":
  {
    COSE_Key: {
      "kty": "symmetric",
      "kid": h'...',
      "k": b64'jb3yjn... #[verifier]
    }
  }
  TOC:{
    as_time: '..',
    nonce: 'rs-nonce..',
  }
  tag: '..' #TOC tag
}

```

Figure 6: Example Access-Response message sent from AS to C with detailed CWT params and payload info

4.7. C->RS : Resource-Request

Once C receives the Access-Response from AS, C can construct a token (Tk) which will demonstrate that C has got the sufficient authorization to access resources (R) in RS.

A new Token (Tk) MUST be attached to each RES-REQ sent to RS by C. If payload data are included, then C should encrypt them using the verifier as key and optionally it can include an Authentication Hash (AuthHash= Hash(verifier+C_nonce)). PAT profile provides necessary recommendations by using AEAD (e.g., chach20/poly1305) algorithm.

o As an example if C performs:

- * A CoAP GET() without payload. In this case, the request from C MAY be sent un-encrypted since it does not include confidential data, but the response from RS with payload MUST be always encrypted and only the valid C MUST be able to decrypt it.
- * A CoAP POST(), a CoAP PUT() or a CoAP DELETE() request with payload MUST be protected and encrypted by using the AEAD algorithm specified in the Access Token (AT). PAT profile proposes to use ChaCha20-Poly1305-AEAD authenticated encryption mechanism, while using the Verifier (PoP key) as the key and a nonce. The AuthHash MAY be protected by using it as Additional Authentication Data (AAD) in the AEAD algorithm.

The RS MUST implement /authz-info endpoint to allow any Client to transfer the token (Tk) as described in [I-D.ietf-ace-oauth-authz]. The Resource-Request message with valid Token (Tk) shall be constructed from AT by C and it should be sent to RS in the following way:

- o Figure 7 shows the example of Client Resource-Request:

```
Request:
Content-Type: application/cose+cbor+pat;
              pat-type="AuthReq";
Message:
{  CoAP request: GET/POST/PUT/DELETE
   Uri-Host "coap://rs.example.com"
   uri-path: /authz-info
   payload:
   {
     Token:
     {
       Access Token(AT), # Tk encapsulates the AT from AS
       AuthHash=Hash(verifier+nonce), #optional for GET
       nonce,
       #Chach20/Poly1305(Verifier,nonce,
         AAD=AuthHash, payload)
       Payload:{
         # if exist
       }
     }
   }
}
```

Figure 7: RES-REQ from C using /authz-info implemented at RS

Figure 7 shows the detailed example of GET RES-REQ to the endpoint /authz-info implemented at RS as described in [I-D.ietf-ace-oauth-

authz]. This option enables the C to transport the token (Tk) to the RS. After receiving the request, RS verifies the token (Tk): RS can construct its own version of verifier or PoP-key by performing $G(K, AT)$ from the access-token (AT); and RS checks whether $AuthHash = Hash(verifier + nonce)$ is valid or not. If Tk and AuthHash are valid, then RS sends an encrypted response using the verifier (PoP key).

- o Figure 8 shows the GET request from C to RS described in [I-D.ietf-ace-oauth-authz], with `pat-type="AuthReq"`.

```
Header: GET
Content-Type: application/cose+cbor+pat;
              pat-type="AuthReq";
Uri-Host: "coap://rs.example.com"
Uri-Path: /authz-info
Payload:
{ token: {
  "access token": .. {
    "aud": "tempSensor"
    "scp": "read"
    ... #CWT omitted for brevity.
  }
  "nonce": ..
  "AuthHash": .. #[AuthHash=hash(verifier+nonce)]
}
TOC:{
  time:'as-time',
  nonce:'rs-nonce',# rs-nonce from RS TOC object
} tag: '..' #TOC tag
}
```

Figure 8: Example of valid GET RES-REQ from C to RS including time-sync using endpoint /authz-info.

The C performs a GET request to "tempSensor" using CWT claim "aud", and together C also transfers the Token (Tk) to the RS. PAT allows performing both RES-REQ and transferring authorization information in RES-REQ. In the next example we show how to perform a resource request if the C performs a POST request with encrypted payload information.

- o Figure 9 shows an example of POST Resource-Request from C to RS described in [I-D.ietf-ace-oauth-authz], with `pat-type="AuthReq"`.

```

Header: POST (Code=0.02)
Content-Type: application/cose+cbor+pat;
              cose-type="encrypt0";
              "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /authz-info
Payload:
{# COSE
  token: {
    "access token": .. {
      "aud": "firmwareUpd"
      "scp": "write"
      ... CWT omitted for brevity,
    }
    "nonce": .. # nonce
    "AuthHash": .. #[AuthHash=hash(verifier+nonce)]
    TOC:{
      time:'as-time',
      nonce:'rs-nonce', # rs-nonce from RS TIC
    } tag: '..' #TOC tag
  }
# COSE_Encrypt0 + COSE_MAC0 Protected
  ciphertext:{
    #Chacha20/Poly1305 AEAD payload using
    # key=verifier,
    # nonce=..,
    # AAD=AuthHash
  },
  tag: ..
}

```

Figure 9: Example of valid POST request from C to RS

Figure 9 shows the POST Resource-Request from C to RS where the Uri-Path "/authz-info" allows the authorized client to perform firmware upgrade on the RS using the CWT claim "aud:firmwareUpd". PAT recommends protecting sensitive information such as the payload using AEAD algorithm (chacha20/poly1305). The client should use Verifier or PoP key as the key, a nonce, and AuthHash as AAD.

4.8. RS->C : Resource-Response

When the RES-REQ with a token (Tk) arrives from C to RS, RS MUST evaluate the resource request and the token (Tk) in the following order:

- o Step 0: Check whether the contents of Tk are derived from an access-token (AT) or not.

- o Step1: If Tk contains the access-token (AT) from AS, extract AT. Extract nonce and Authentication Hash (AuthHash) from the request message.
 - * Step1.1: (If available) Verify the freshness of the sequence number (cti) in the access token presented by AS.
 - * Step1.2: Generate the verifier by computing $G(K, \text{access token})$ where K is the shared key between AS and RS.
 - * Step1.3: Compute a verification hash as $\text{Hash}(\text{verifier} + \text{nonce})$ and compare the result with AuthHash for correctness.
 - * Step1.4: Check if the access token has valid CWT parameters such as "aud", "scp", "exp", "nbf", etc for the requested resource or action to be performed.
 - * Step1.5: (IF available) Synchronize RS internal clock using TOC object as described in [I.D-draft-navas-ace-secure-time-synchronization].
- o Step2: If the token is valid, RS should create a temporary internal state as shown in table 1 below with details of CWT claims "cti", "exp", "scp", and the verifier (PoP key).

The RS internal state table which is shown in Table1 also includes "next cti". The next cti (cti x) value is computed as the Hash of previous cti (cti x-1) and the verifier. The purpose of this is explained in the section Section 4.9.

Verifier	cti_x-1	exp	scp	next cti (cti_x)
$G(k, AT)$	cti_x0= cti of AT	of AT	of AT	cti_x1= hash(cti_x0, Verifier)

Table 1: RS Internal state table of access-tokens and RS_nonce

- o Step 3: If the token is valid, then RS decrypts the payload from the client (if exist) Verifier (PoP key).
- o Step 4: After that, RS prepares the response and encrypts the payload with a fresh nonce, PoP key. Only the Client (C) with a valid key (the Verifier) can decrypt the payload:

4.8.1. RS Response-codes to C RES-REQ:

- o If the token (Tk) is valid -- as discussed above --, then RS MUST respond with payload-data as described above with the appropriate response code as described in [RFC7252]. For example, to a POST request with 2.01 (created) or 2.04 (changed).
- o If the token (Tk) is invalid, then RS MUST respond with code 4.01 (Unauthorized)
- o If the token (Tk) is valid but does not match the "aud" or resource C is requesting for then RS MUST respond with code 4.03 (Forbidden)

4.9. Construction of Derived-Tokens (DT)

The objectives to create Derived-Tokens (DT) are:

- o To produce Unlinkable Tokens (Tk). It is not efficient for the client to request a new access-token (AT) from AS everytime. Also, if C uses the same access-token (AT) from AS, the identity of the client can be identified via the AT CWT claim "cti" (token identity).
- o To reduce token (Tk) size (efficiency in transport) that the client must send to RS /authz-info in every resource request.
- o To create tokens (Tk) that may have limited access to protected-resources -- fine-grained resource access tokens -- from the original access-tokens (AT) that could grant more privileges to protected-resources on RS. For example, an access-token (AT) could provide permissions to access all protected-resources on RS via CWT claims audience "aud" and scope "scp". The client could derive a Token (Tk) providing access to a reduced set of protected-resources available on RS from the initial AT.

4.9.1. C->RS: Resource-Request via DT

The Client receives an encrypted response from RS after its first RES-REQ with the access-token (AT) from AS.

The Client creates a new Derived-Token(DT) using CWT claims as described below. In order to minimize the data size, we use only the claims which are required.

- o Client MAY prepare a DT with a subset of scope "scp" operations that the client received from the initial Access-Token (AT). It creates the first derived "cti_x1" by Hash("cti_x0 + verifier")

from the CWT claim "cti" of the original access-token (AT). The subsequent derivation of "cti_x" can be performed by a generic function "cti_x = Hash(cti_x-1 + verifier)". Note that the derived-token (DT) MUST include all the necessary CWT claims such as "cti_x", "aud", "exp", "scp". All other CWT claims are optional.

- o Client creates the AuthHash=(verifier+nonce).
- o Client prepares encrypted content using verifier as the key -- if there is any payload --.
- o Note: in the Additional Authenticated data (AAD), the C includes AuthHash and the derived-token (DT), so that the payload cannot be misused/exchanged with another RES-REQ or nonce.

```

Header: POST (Code=0.02)
Content-Type: application/cbor+cwt+cose++pat;
              cose-type="encrypt0";
              "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /firmware
Payload:
{# COSE
  token: {derived-token(DT):
    "aud": "firmwareUpd",
    "exp": ..
    "scp": "write",
    "cti": Hash(cti_x+verifier)
    # cti_x=Hash(cti_x-1+verifier).
  }
  "nonce": .. # new nonce
  "AuthHash": h'bfa03.. #[Hash=(verifier+nonce)]
# COSE_Encrypt0 + COSE_MAC0 Protected
  ciphertext:{
    #Chacha20/Poly1305 AEAD payload using
    # key=verifier,
    # nonce=..,
    # AAD=AuthHash,DT
    h'....omitted for brevity
  },
  tag: h'... omitted for brevity
}
```

Figure 12: Example of valid Resource-Request from C to RS using a derived-token(DT)

4.9.2. RS->C : Resource-Response to DT

After receiving the Token (Tk) which encapsulates the derived Token (DT) from C, RS performs the following Steps. If any of them fails, then RS must send an Unauthorized response to C, and C must use the first AT, which was received from the AS, or request a new AT based on the resource owner (RO) configuration:

- o RS extracts CWT claim cti (cti_x) from the Derived-Token (DT) and checks if it exists in its internal state table. If RS finds the cti_x, then RS uses the corresponding verifier, "cti_x-1", "exp", and "scp" to perform the validation of next steps.
- o RS checks that $cti_x = \text{Hash}(cti_x-1 + \text{verifier})$
- o RS checks that $\text{AuthHash} == \text{Hash}(\text{verifier} + \text{nonce})$
- o RS checks that the permissions are valid using "scp" and expiration time "exp"
- o RS updates the new cti_x-1, cti_x in its internal state table
- o RS creates an encrypted response to be sent to C with a payload including payload-data.

msg#	Verifier (V)	cti_x-1	exp	scp	cti_x= Hash(cti_x-1+V)
0	G(K,AT)	0x00	of AT	of AT	0xAB = Hash(0x00+V)
1 (upd)	G(k,AT)	0xAB	of AT	of AT	0xFF = Hash(0xAB+V)

Table 2: RS updating only two parameters in its internal stating table 1

The Table 2 shows the RS internal state table with an example.

5. Security Considerations

TBD

5.1. Privacy Considerations

The CoAP messaging layer parameters such as token and message-id can be used for matching a specific request and response. TBD

6. IANA Considerations

TBD

7. References

7.1. Normative References

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

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

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

[RFC7539] Y. Nir and A. Langley: ChaCha20 and Poly1305 for IETF Protocols, RFC7539, May 2015

[I-D.ietf-ace-actors] Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-0 (work in progress), March 2017.

[I-D.ietf-oauth-pop-architecture] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[I.D-draft-navas-ace-secure-time-synchronization] Navas, G., Selander, G., Seitz, L., "Lightweight Authenticated Time (LATE) Synchronization Protocol", draft-navas-ace-secure-time-synchronization-00 (work in progress), October 2016.

7.2. Informative References

[KoMa2014] Kohnstamm, J. and Madhub, D., "Mauritius Declaration on the Internet of Things", 36th International Conference of Data Protection and Privacy Commissioners, October 2014.

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

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

[I.D-draft-gerdes-ace-dtls-authorize] Gerdes, S., Begmann, O., Bormann, C., Selander, G., Seitz, L. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE), draft-gerdes-ace-dtls-authorize-01, March 2017.

[I-D.ietf-ace-cbor-web-token] Jones, M., Tschofenig, H., Erdtman, S., CBOR Web Token (CWT), draft-ietf-ace-cbor-web-token-05 (work in progress), June 2017..

8. Acknowledgement

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). The authors thank Ludwig Seitz for reviewing the previous version of the draft.

8.1. 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>>.

Appendix A. ACE profile Registration

TBD

ACE profile template	PAT
Profile name	TBD
Profile Description	TBD
Profile ID	TBD

Table2: ACE profile registration template

Authors' Addresses

Jorge Cuellar
Siemens AG
Otto-Hahn-Ring 6
Munich, Germany 81739

Email: jorge.cuellar@siemens.com

Prabhakaran Kasinathan
Siemens AG
Otto-Hahn-Ring 6
Munich, Germany 81739

Email: prabhakaran.kasinathan@siemens.com

Daniel Calvo
Atos Research and Innovation
Poligono Industrial Candina
Santander, Spain 39011

Email: daniel.calvo@atos.net

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

S. Gerdes
Universitaet Bremen TZI
J. Cuellar
Siemens AG
O. Bergmann
Universitaet Bremen TZI
October 19, 2015

Solutions for the authorization in constrained environments
draft-cuellar-ace-solutions-00

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. 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

3.9.3. Protection Requirements	9
3.10. Security Association between C and S	10
3.10.1. Effect	10
3.10.2. Actors	10
3.10.3. Protection Requirements	10
3.11. Authorized Resource Request Message	10
3.11.1. Effect	10
3.11.2. Actors	10
3.11.3. Protection Requirements	11
3.12. Resource Response Message	11
3.12.1. Effect	11
3.12.2. Actors	11
3.12.3. Protection Requirements	11
3.13. Server-Initiated Ticket Request Messages	11
4. Content Format	11
5. Security Considerations	11
6. IANA Considerations	12
7. Acknowledgements	12
8. Informative References	12
Authors' Addresses	12

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

- o 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.
- o 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

- o C -> S
- or, optionally,
- o 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

- o 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

- o Integrity and Authenticity (CAM can validate that the message stems from C)
- o 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

- o SAM knows which actions on which resources are requested by CAM.
- o SAM can determine permissions for CAM.
- o SAM can generate an access ticket for C, which can be later used by C to demonstrate to S its authorization status.
- o 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

- o CAM -> SAM

3.5.3. Protection Requirements

- o Integrity and authenticity (SAM can validate that the message stems from CAM)
- o 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

- o CAM received the Server Authorization Information (SAI)
- o CAM received the verifier for C

- o CAM can validate the origin of the ticket for C

3.6.2. Actors

- o SAM -> CAM

3.6.3. Protection Requirements

- o Confidentiality (SAM, CAM) (+ Integrity (implicit, the ticket already is integrity-protected))
- o 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

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

3.7.2. Actors

- o CAM -> C

3.7.3. Protection Requirements

- o 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

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

3.8.2. Actors

- o CAM -> C

3.8.3. Protection Requirements

- o Integrity: attackers must not be able to manipulate the CAI.
- o Confidentiality (optional): in some cases, principals might not want others to gain knowledge of the CAI.
- o 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

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

3.9.2. Actors

- o 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

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

3.10.2. Actors

- o C, S

3.10.3. Protection Requirements

- o Integrity: Attackers must not be able to update the authorization information stored at S and C.
- o 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

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

3.11.2. Actors

- o C -> S

3.11.3. Protection Requirements

- o Integrity
- o 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

- o C receives the requested service from S.

3.12.2. Actors

- o S -> C

3.12.3. Protection Requirements

- o Integrity
- o 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]

Gerdes, S., "Server-Initiated Ticket Request", draft-gerdes-ace-sitr-00 (work in progress), October 2015.

[I-D.ietf-ace-actors]

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

[I-D.ietf-ace-usecases]

Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-09 (work in progress), October 2015.

[I-D.ietf-cose-msg]

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

Authors' Addresses

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

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

Jorge Cuellar
Siemens AG
CT RTC ITS

Email: jorge.cuellar@siemens.com

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

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

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

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI
October 19, 2015

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

Abstract

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

Status of This Memo

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. Listing Authorization Manager Information in a Resource Directory	30
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

10.4. Implicit Authorization	35
11. Specific Usage Scenarios	36
11.1. Combined Authorization Manager and Client	36
11.1.1. Creating the Ticket Request Message	36
11.1.2. Processing the Ticket Grant Message	37
11.2. Combined Client Authorization Manager and Server Authorization Manager	37
11.2.1. Processing the Access Request Message	38
11.2.2. Creating the Ticket Transfer Message	38
11.3. Combined Server Authorization Manager and Server	38
12. Security Considerations	39
13. IANA Considerations	39
13.1. DTLS PSK Key Generation Methods	40
13.2. dcaf+cbor Media Type Registration	40
13.3. CoAP Content Format Registration	41
14. Acknowledgements	41
15. References	42
15.1. Normative References	42
15.2. Informative References	43
Appendix A. CDDL Specification	44
Authors' Addresses	45

1. Introduction

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

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

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

As fine-grained authorization is not always needed on constrained devices, DCAF supports an implicit authorization mode where no authorization information is exchanged.

The main goals of DCAF are the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] between two nodes and to securely transmit authorization tickets.

1.1. Features

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

1.2. Terminology

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

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

1.2.1. Actors

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

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

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

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

Authorization Manager (AM): An entity that is either a SAM or a CAM.

Client Overseeing Principal (COP): The principal that is in charge of the Client and controls permissions concerning authorized representations of a CoAP resource.

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

1.2.2. Other Terms

Resource (R): A CoAP resource.

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

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

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

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

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

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

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

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

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

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

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

2. System Overview

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

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

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

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

- o AM must have enough stable storage (such as disk space) to store the necessary number of credentials (matching the number of Clients and Servers).
- o AM must possess means for user interaction, for example directly or indirectly connected input/output devices such as keyboard and

display, to allow for configuration of authorization information by the respective Principal.

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

3. Protocol

The DCAF protocol comprises three parts:

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

3.1. Overview

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

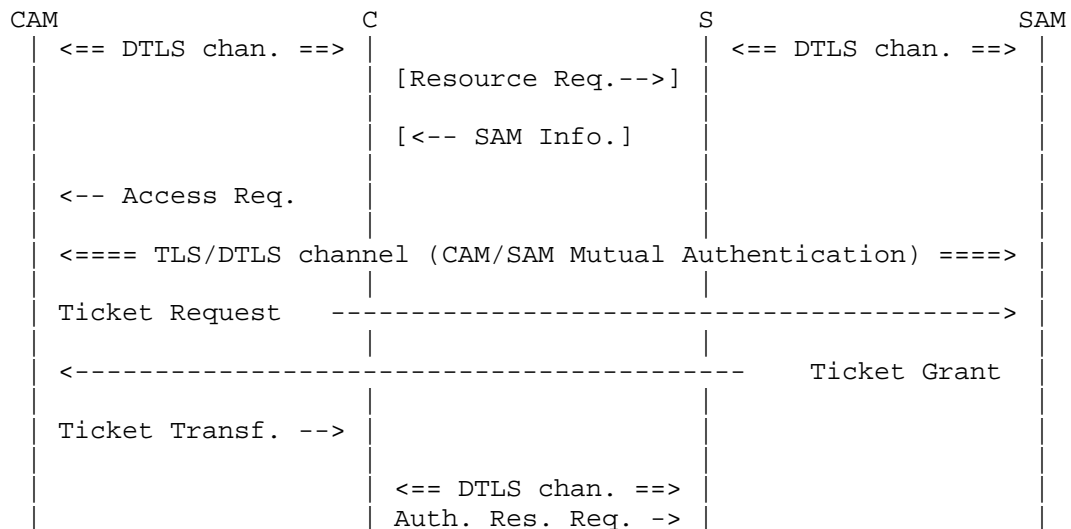


Figure 1: Protocol Overview

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

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

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

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

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

This document uses Concise Binary Object Representation (CBOR, [RFC7049]) to express authorization information as set of attributes passed in CoAP payloads. Notation and encoding options are discussed in Section 5. A formal specification of the DCAF message format is given in Appendix A.

3.2. Unauthorized Resource Request Message

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

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

- o S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

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

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

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

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

3.3. SAM Information Message

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

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

The message MAY also contain a timestamp generated by S.

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

4.01 Unauthorized

Content-Format: application/dcaf+cbor

```
{SAM: "coaps://sam.example.com/authorize", TS: 168537,
  A: [ TBD1, ct_cose_msg ] }
```

Figure 2: SAM Information Payload Example

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

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

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

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

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

Figure 3: SAM Information Payload Example encoded in CBOR

3.3.1. Piggybacked Protected Content

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

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

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

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

3.4. Access Request

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

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

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

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

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

1. Contact information for the SAM to use.
2. An absolute URI of the resource that C wants to access.

3. The actions that C wants to perform on the resource.
4. Any time stamp generated by S.

An example Access Request from C to CAM is depicted in Figure 4. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

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

Figure 4: Access Request Message Example

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

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

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

3.5. Ticket Request Message

When CAM receives an Access Request message from C and COP specified authorization policies for C, CAM MUST check if the requested actions are allowed according to these policies. If all requested actions are forbidden, CAM MUST send a 4.03 response.

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

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

If CAM does not send a cached response, it checks whether the request payload is of type "application/dcaf+cbor" and contains at least the fields SAM and SAI. CAM MUST respond with 4.00 (Bad Request) if the type is "application/dcaf+cbor" and any of these fields is missing or does not conform to the format described in Section 5.

If the payload is correct, CAM creates a Ticket Request message from the Access Request received from C as follows:

1. The destination of the Ticket Request message is derived from the "SAM" field that is specified in the Access Request message payload (for example, if the Access Request contained 'SAM: "coaps://sam.example.com/authz"', the destination of the Ticket Request message is sam.example.com).
2. The request method is POST.
3. The request URI is constructed from the SAM field received in the Access Request message payload.
4. The payload is copied from the Access Request sent by C.

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

3.6. Ticket Grant Message

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

SAM decides whether or not access is granted to the requested resource and then creates a Ticket Grant message that reflects the result. To grant access to the requested resource, SAM creates an access ticket comprised of a Face and the Client Information as described in Section 4.

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

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

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

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

```

2.05 Content
Content-Format: application/dcaf+cbor
Max-Age: 86400
{ F: {
    SAI: [ "/s/tempC", 7 ],
    TS: 0("2013-07-10T10:04:12.391"),
    L: 86400,
    G: hmac_sha256
  },
  V: h'f89947160c73601c7a65cb5e08812026
    6d0f0565160e3ff7d3907441cdf44cc9'
}
```

Figure 5: Example Ticket Grant Message

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

```

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

Figure 6: Example Ticket Grant Message With Reject

3.7. Ticket Transfer Message

A Ticket Transfer message delivers the access information sent by SAM in a Ticket Grant message to the requesting client C. The Ticket Transfer message is the response to the Access Request message sent from C to CAM and includes the ticket data from SAM contained in the Ticket Grant message.

The Authorization Information provided by SAM in the Ticket Grant Message may grant more permissions than C has requested. The authorization policies of COP and ROP may differ: COP might want restrict the resources C is allowed to access, and the actions that C is allowed to perform on the resource.

If COP defined authorization policies that concern the requested actions, CAM MUST add Authorization Information for C (CAI) to the CI that reflect those policies. Since C and CAM use a DTLS channel for communication, the 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
    6d0f0565160e3ff7d3907441cdf44cc9'
  CAI: [ "/s/tempC", 1 ],
  TS: 0("2013-07-10T10:04:12.855"),
  L: 86400
}
```

Figure 7: Example Ticket Transfer Message

3.8. DTLS Channel Setup Between C and S

When C receives a Ticket Transfer message, it checks if the payload contains a face and a Client Information. With this information C can initiate establishment of a new DTLS channel with S. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], with the following additional requirements:

1. C sets the `psk_identity` field of the ClientKeyExchange message to the ticket Face received in the Ticket Transfer message.
2. C uses the ticket Verifier as PSK when constructing the premaster secret.

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

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

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

3.9. Authorized Resource Request Message

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

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

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

1. The message was received on a secure channel that has been established using the procedure defined in Section 3.8.
2. The authorization information tied to the secure channel is valid.
3. The request is destined for S.
4. The resource URI specified in the request is covered by the authorization information.
5. The request method is an authorized action on the resource with respect to the authorization information.

Note that the authorization information is not restricted to a single resource URI. For example, role-based authorization can be used to authorize a collection of semantically connected resources simultaneously. Implicit authorization also provides access rights to authenticated clients for all actions on all resources that S offers. As a result, C can use the same DTLS channel not only for subsequent requests for the same resource (e.g. for block-wise transfer as defined in [I-D.ietf-core-block] or refreshing observe-relationships [RFC7641]) but also for requests to distinct resources.

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

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and
2. with response code 4.05 (Method Not Allowed) when the resource URI specified in the request covered by the authorization information but not the requested action.

Since SAM may limit the set of requested actions in its Ticket Grant message, C cannot know a priori if an Authorized Resource Request will succeed. If C repeatedly gets SAM Information messages as response to its requests, it SHOULD NOT send new Access Requests to CAM.

3.10. Dynamic Update of Authorization Information

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

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

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

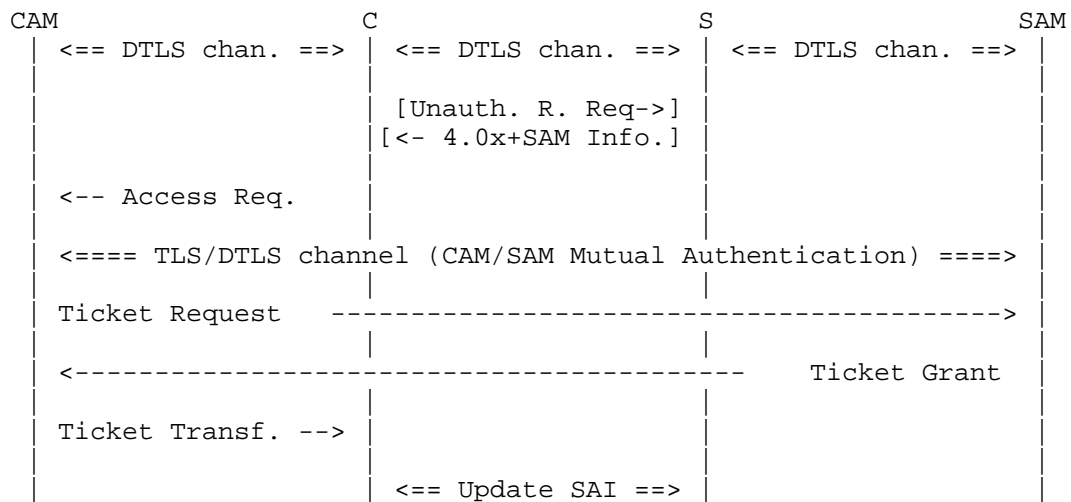


Figure 8: Overview of Dynamic Update Operation

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

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

3.10.1. Handling of Ticket Transfer Messages

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

After successful completion of the DTLS handshake S updates the existing SAM Authorization Information for C according to the contents of the ticket Face.

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

4. Ticket

Access tokens in DCAF are tickets that consist of two parts, namely the Face and the Client Information (CI). SAM generates the ticket Face for S and the verifier that corresponds to the ticket Face for C. The verifier is included in the CI.

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

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

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

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

4.1. Face

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

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

Optionally, Face MAY also contain:

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

- o A SAI identifier (optional)

S MUST verify the integrity of Face, i.e. the information contained in Face stems from SAM and was not manipulated by anyone else. The integrity of Face can be ensured by various means. Face may be encrypted by SAM with a key it shares with S. Alternatively, S can use a mechanism to generate the DTLS PSK which includes Face. S generates the key from the Face it received. The correct key can only be calculated with the correct Face (refer to Section 6 for details).

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

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

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

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

4.2. Client Information

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

- o The Verifier generated by SAM

CI MAY additionally contain:

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

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

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

4.3. Revocation

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

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

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

Table 1: DCAF field identifiers encoded in CBOR

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

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

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

- CAI: CAM Authorization Information. A data structure used to convey authorization information from CAM to C. It describes the C's permissions for S according to CAM, e.g., which actions C is allowed to perform on an R of S. The CAI attribute contains an AIF object as defined in [I-D.bormann-core-ace-aif].
- A: Accepted content formats. An array of numeric content formats from the CoAP Content-Formats registry (c.f. Section 12.3 of [RFC7252]).
- D: Protected Data. A binary string containing data that may be encrypted.
- E: Encrypted Ticket Face. A binary string containing an encrypted ticket Face.
- K: Key. A string that identifies the shared key between S and SAM that can be used to decrypt the contents of E. If the attribute E is present and no attribute K has been specified, the default is to use the current session key for the secured channel between S and SAM.
- TS: Time Stamp. A time stamp that indicates the instant when the access ticket request was formed. This attribute can be used by the Server in an SAM Information message to convey a time stamp in its local time scale (e.g. when it does not have a real time clock with synchronized global time). When the attribute's value is encoded as a string, it MUST contain a valid UTC timestamp without time zone information. When encoded as integer, TS contains a system timestamp relative to the local time scale of its generator, usually S.
- L: Lifetime. When included in a ticket face, the contents of the L parameter denote the lifetime of the ticket. In combination with the protected data field D, this parameter denotes the lifetime of the protected data. When encoded as a string, L MUST denote the ticket's expiry time as a valid UTC timestamp without time zone information. When encoded as an integer, L MUST denote the ticket's validity period in seconds relative to TS.
- N: Nonce. An initialization vector used in combination with piggybacked protected content.
- G: DTLS PSK Generation Method. A numeric identifier for the method that S MUST use to derive the DTLS PSK from the ticket Face. This attribute MUST NOT be used when attribute V is present within the contents of F. This specification uses symbolic identifiers for improved readability. The corresponding numeric values encoded in

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

- F: Ticket Face. An object containing the fields SAI, TS, and optionally G, L and V.
- V: Ticket Verifier. A binary string containing the shared secret between C and S.

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

Table 2: CBOR encoding for DTLS PSK Key Generation Methods

5.1. Examples

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

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

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

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

The Verifier V contains a shared secret to be used as DTLS PSK between C and S.

```

HTTP/1.1 200 OK
Content-Type: application/dcaf+cbor
{
  F: {
    SAI: [ "/s/tempC", 1 ],
    TS: 2938749,
    L: 3600,
    G: hmac_sha256
  },
  V: h'48ae5a81b87241d81618f56cab0b65ec
    441202f81faabbel10075b20cb57fa939'
}

```

The Face may be encrypted as illustrated in the following example. Here, the field E carries an encrypted Face data structure that contains the same information as the previous example, and an additional Verifier. Encryption was done with a secret shared by SAM and S. (This example uses AES128_CCM with the secret { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f } and S's timestamp { 0x00, 0x2C, 0xD7, 0x7D } as nonce.) Line breaks have been inserted to improve readability.

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

```

{
  E: h'2e75eeae01b831e0b65c2976e06d90f4
    82135bec5efef3be3d31520b2fa8c6fb
    f572f817203bf7a0940bb6183697567c
    e291b03e9fca5e9cbdfa7e560322d4ed
    3a659f44a542e55331a1a9f43d7f',
  K: "key0",
  V: h'48ae5a81b87241d81618f56cab0b65ec
    441202f81faabbel10075b20cb57fa939'
}

```

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

```
{
  F: {
    SAI: [ "/s/tempC", 1 ],
    TS: 2938749,
    L: 3600,
    G: hmac_sha256
  },
  V: h'48ae5a81b87241d81618f56cab0b65ec
    441202f81faabbel0075b20cb57fa939'
}
```

6. DTLS PSK Generation Methods

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

6.1. DTLS PSK Transfer

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

The DTLS PSK transfer method is defined as follows:

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

6.2. Distributed Key Derivation

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

The distributed key derivation method is defined as follows:

- o SAM and S both generate the DTLS PSK using the information included in Face. They use an HMAC algorithm on Face with a shared key $K(\text{SAM}, S)$. The result serves as the DTLS PSK. How SAM and S exchange $K(\text{SAM}, S)$ is not in the scope of this document. They MAY use their preshared key as $K(\text{SAM}, S)$. How SAM and S negotiate the used HMAC algorithm is also not in the scope of this document. They MAY however use the HMAC algorithm they use for their DTLS connection.
- o SAM MUST include a representation of the DTLS PSK in the Verifier.
- o As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- o SAM MUST NOT include a representation of the DTLS PSK in Face.
- o SAM MUST NOT encrypt Face.

7. Authorization Configuration

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

8. Trust Relationships

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

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

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

1. SAM and CAM have means to mutually authenticate each other (e.g., they might have a certificate of the other party or a PKI in which it is included)
2. If SAM requires information about the client from SAM, e.g. if SAM only 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 SAM with some potentially confidential information, and trusts that SAM correctly represents S, because CAM trusts SAM.

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

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

CAM <-----> SAM

```

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

```

C S

9. Listing Authorization Manager Information in a Resource Directory

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

9.1. The "auth-request" Link Relation

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

The Content-Format "application/dcaf+cbor with numeric identifier TBD1" defined in this specification MAY be used to express access requests and their responses.

The following example shows the web link used by CAM in this document to relay incoming Authorization Request messages to SAM. (Whitespace is included only for readability.)

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

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

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

10. Examples

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

10.1. Access Granted

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

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

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

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

CAM --> SAM [Mid=23146]
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 4]
}

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

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



```
}

C --> S
ClientHello (TLS_PSK_WITH_AES_128_CCM_8)

C <-- S
ServerHello (TLS_PSK_WITH_AES_128_CCM_8)
ServerHelloDone

C --> S
ClientKeyExchange
  psk_identity=0xa301826c612f73776974636832393431
                0x0505c077323031332d30372d30345432
                0x303a31373a33382e3030320700

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

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

```
C <-- S
ChangeCipherSpec
Finished
```

10.2. Access Denied

This example shows a denied Authorization request for the DELETE operation.

```
C --> S
DELETE a/switch2941

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

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

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

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

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

10.3. Access Restricted

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

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

```
CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor
{ F: {
  SAI: ["a/switch2941", 5],
  TS: 0("2013-07-04T21:33:11.930"),
  G: hmac_sha256
},
V: h'c7b5774f2ddcbd548f4ad74b30a1b2e5
    b6b04e66a9995edd2545e5a06216c53d'
}
```

10.4. Implicit Authorization

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

```
CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 3]
}
```

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

11. Specific Usage Scenarios

The general DCAF 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.

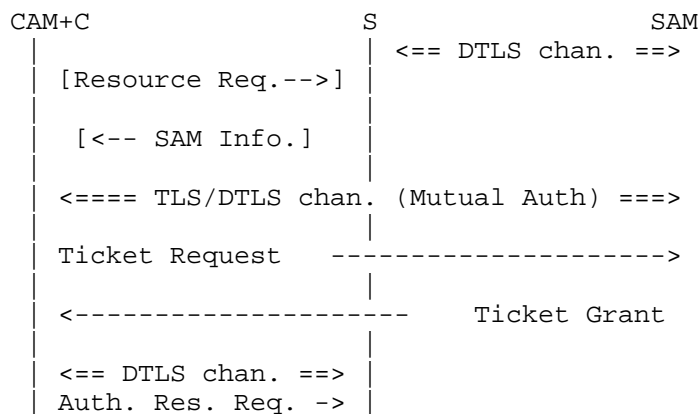


Figure 9: Combined Client Authorization Manager and Client

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 `ticketFace` 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.

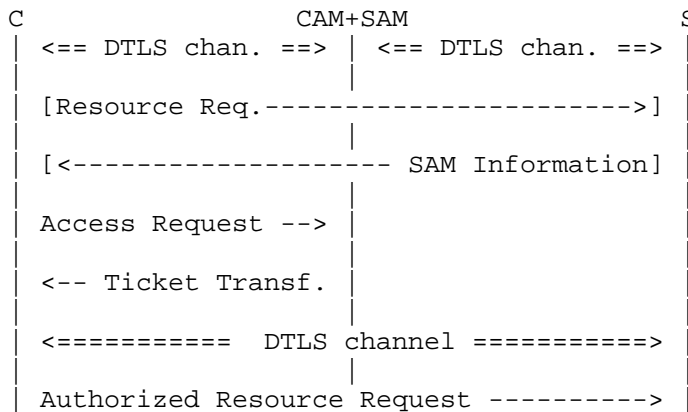


Figure 10: Combined Client Authorization Manager and Server
Authorization Manager

11.2.1. Processing the Access Request Message

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

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

11.2.2. Creating the Ticket Transfer Message

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

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

11.3. Combined Server Authorization Manager and Server

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

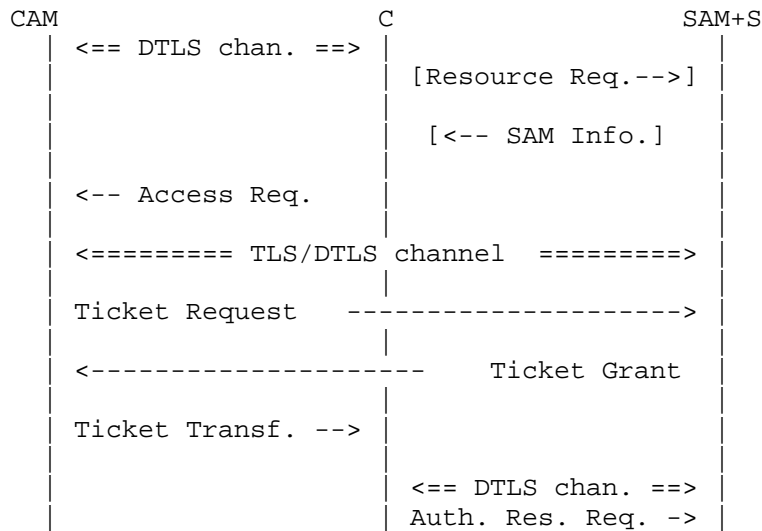


Figure 11: Combined Server Authorization Manager and Server

12. Security Considerations

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

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

13. IANA Considerations

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

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

13.1. DTLS PSK Key Generation Methods

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

Initial entries in this sub-registry are as follows:

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

Table 3: DTLS PSK Key Generation Methods

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

(TBD: criteria for expert review.)

13.2. dcaf+cbor Media Type Registration

Type name: application

Subtype name: dcaf+cbor

Required parameters: none

Optional parameters: none

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

Security considerations: TBD

Interoperability considerations: TBD

Published specification: [RFC-XXXX]

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): dcaf

Macintosh file type code(s): none

Person & email address to contact for further information: TBD

Intended usage: COMMON

Restrictions on usage: None

Author: TBD

Change controller: IESG

13.3. CoAP Content Format Registration

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

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

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

14. Acknowledgements

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

15. References

15.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <<http://www.rfc-editor.org/info/rfc4279>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <<http://www.rfc-editor.org/info/rfc5746>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<http://www.rfc-editor.org/info/rfc6838>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.

15.2. Informative References

- [I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-03 (work in progress), July 2015.
- [I-D.gerdes-ace-a2a]
Gerdes, S., "Managing the Authorization to Authorize in the Lifecycle of a Constrained Device", draft-gerdes-ace-a2a-01 (work in progress), September 2015.
- [I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-07 (work in progress), October 2015.
- [I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.
- [I-D.ietf-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 2015.
- [I-D.schmertmann-dice-codtls]
Schmertmann, L., Hartke, K., and C. Bormann, "CoDTLS: DTLS handshakes over CoAP", draft-schmertmann-dice-codtls-01 (work in progress), August 2014.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010,
<<http://www.rfc-editor.org/info/rfc5988>>.

- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <<http://www.rfc-editor.org/info/rfc6655>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<http://www.rfc-editor.org/info/rfc7641>>.
- [bergmann12] Bergmann, O., Gerdes, S., Schaefer, S., Junge, F., and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network", IEEE Wireless Communications and Networking Conference Workshops (WCNCW), April 2012.

Appendix A. CDDL Specification

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

[I-D.greevenbosch-appsawg-cbor-cddl]:

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

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

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

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

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

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

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

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

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

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

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

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

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

Authors' Addresses

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

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

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

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

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

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

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

S. Gerdes
Universitaet Bremen TZI
October 19, 2015

Server-Initiated Ticket Request
draft-gerdes-ace-dcaf-sitr-00

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.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	2
2. Protocol	3
2.1. Overview	3
2.2. CAM Information Message	4
2.3. Server-Initiated Access Request Message	5
2.4. Server-Initiated Ticket Request Message	5
2.5. SI Ticket Grant Message	6
2.6. SI Ticket Transfer Message	7
2.7. CAM Information Response	7
3. Payload Format	8
4. Content Types	8
5. IANA Considerations	8
6. Security Considerations	8
7. Acknowledgments	9
8. References	9
8.1. Normative References	9
8.2. Informative References	9
Author's Address	9

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:

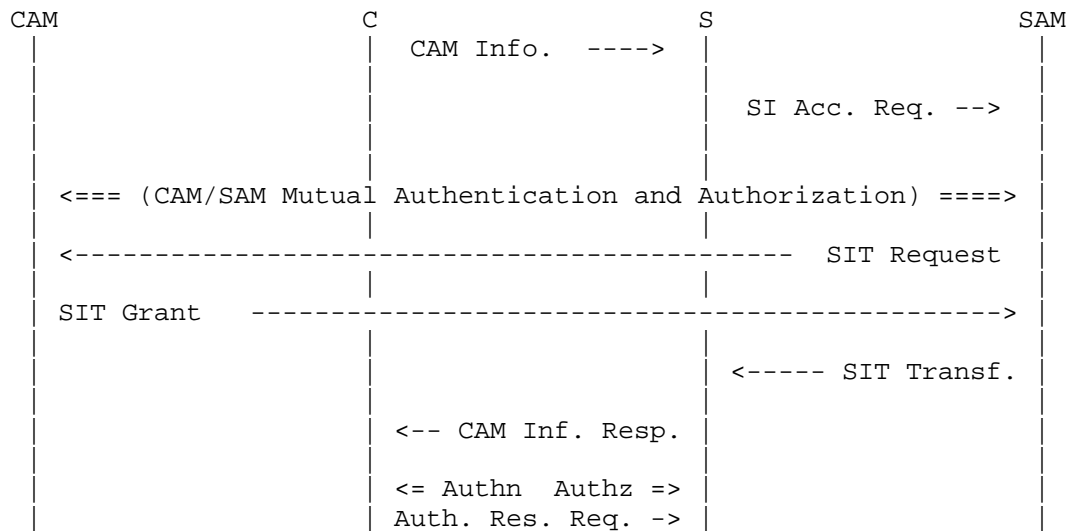


Figure 1: Protocol Overview

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:

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

8.2. Informative References

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

Author's Address

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

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

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

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

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

Abstract

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

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

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

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

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

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

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

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

1.1. Terminology

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

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

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

In addition, this document uses the following terminology:

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

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

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

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

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

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

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

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

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

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

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

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

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

2. Architecture and High-level Problem Statement

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

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

2.1. Elements of an Architecture

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

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

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

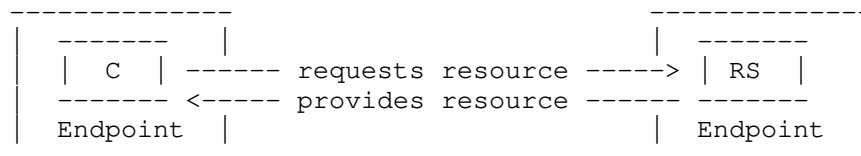


Figure 1: Constrained Level

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

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

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

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

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

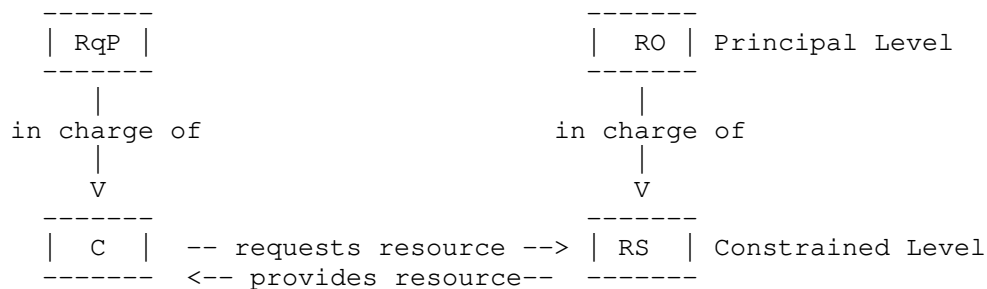


Figure 2: Constrained Level and Principal Level

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

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

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

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

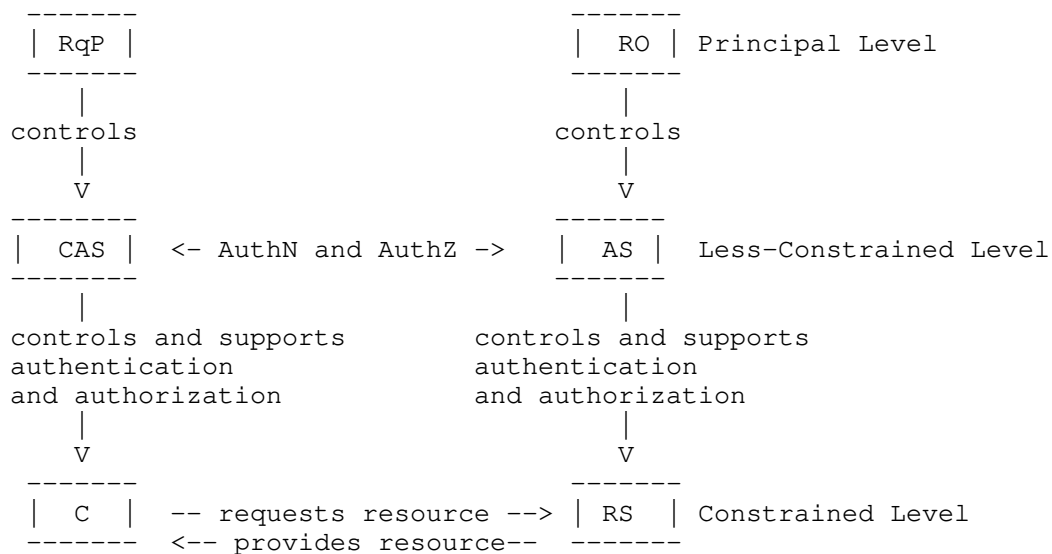


Figure 3: Overall architecture

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

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

2.2. Architecture Variants

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

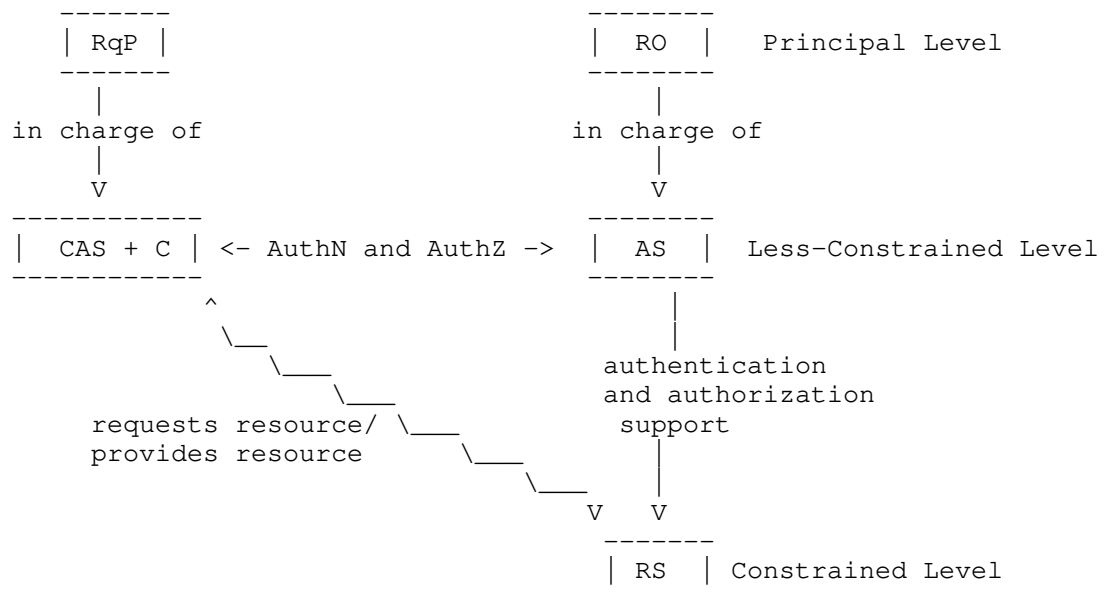


Figure 4: Combined C and CAS

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

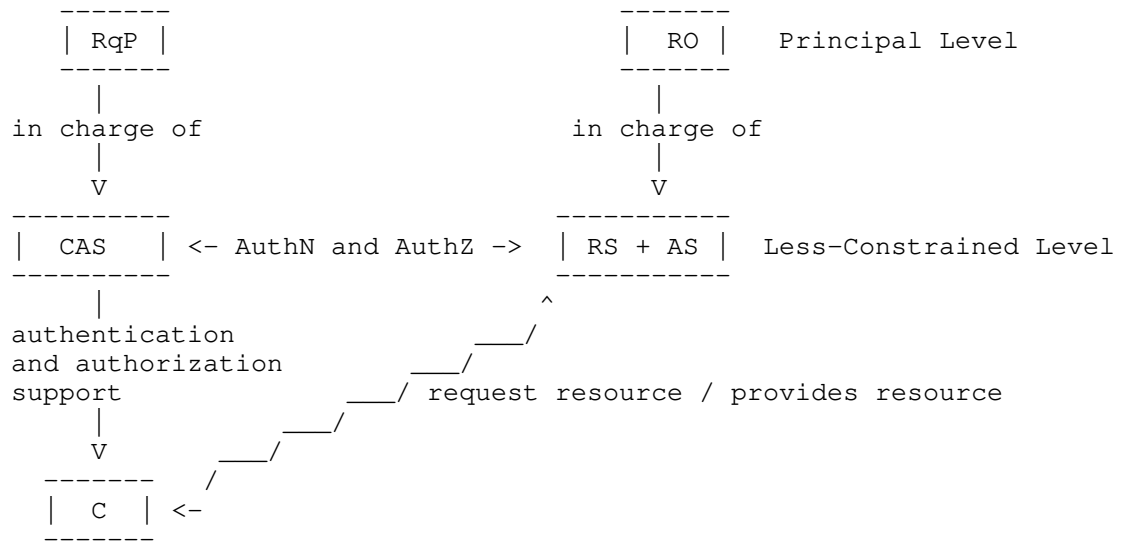


Figure 5: Combined AS and RS

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

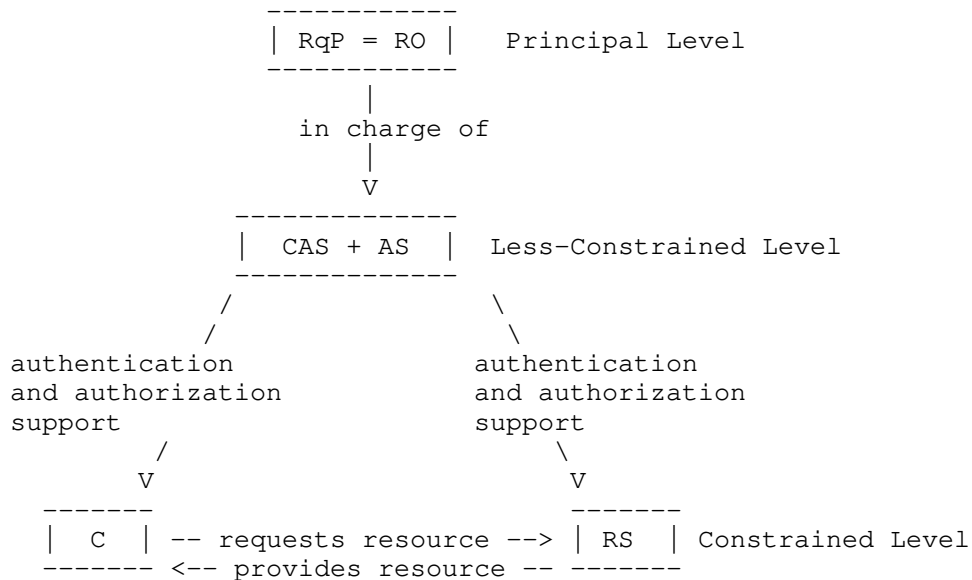


Figure 6: CAS combined with AS

2.3. Information Flows

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

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

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

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

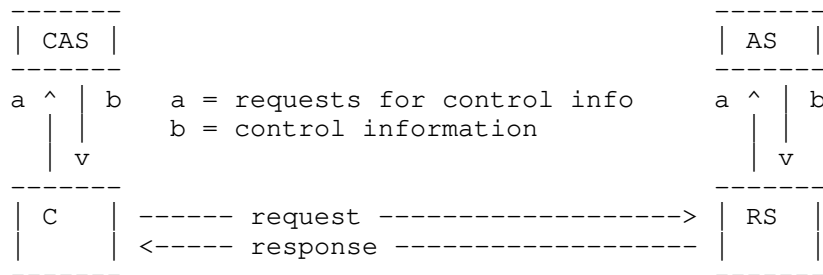


Figure 7: Information flows that need to be protected

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

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

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

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

3. Security Objectives

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

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

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

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

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

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

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

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

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

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

4. Authentication and Authorization

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

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

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

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

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

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

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

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

Table 1: Some granularity levels for authorization

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

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

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

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

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

5. Actors and their Tasks

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

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

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

5.1. Constrained Level Actors

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

C performs the following tasks:

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

RS performs the following tasks:

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

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

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

(See Figure 8.)

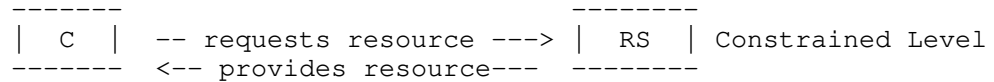


Figure 8: Constrained Level Actors

5.2. Principal Level Actors

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

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

RqP must fulfill the following task:

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

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

RO must fulfill the following task:

- o Configure for RS authorization information for accessing R.

(See Figure 2.)

5.3. Less-Constrained Level Actors

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

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

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

CAS performs the following tasks:

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

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

AS performs the following tasks:

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

6. Kinds of Protocols

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

6.1. Constrained Level Protocols

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

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

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

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

6.1.1. Cross Level Support Protocols

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

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

6.2. Less-Constrained Level Protocols

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

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

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

7. Elements of a Solution

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

7.1. Authorization

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

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

7.2. Authentication

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

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

7.3. Communication Security

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

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

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

7.4. Cryptographic Keys

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

Symmetric vs Asymmetric Keys

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

Key Establishment

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

Revocation and Expiration

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

8. Assumptions and Requirements

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

8.1. Constrained Devices

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

8.2. Server-side Authorization

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

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

8.3. Client-side Authorization Information

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

8.4. Resource Access

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

8.5. Keys and Cipher Suites

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

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

8.6. Network Considerations

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

9. Security Considerations

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

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

9.1. Physical Attacks on Sensor and Actuator Networks

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

The main threats to sensors and actuator networks are:

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

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

9.2. Clocks and Time Measurements

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

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

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

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

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

10. IANA Considerations

This document has no actions for IANA.

11. Informative References

[HUM14delegation]

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

[I-D.hardjono-oauth-umacore]

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

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

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

[I-D.ietf-core-object-security]

Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.

- [I-D.irtf-t2trg-iot-secons]
Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-secons-15 (work in progress), May 2018.
- [REST]
Fielding, R. and R. Taylor, "Principled design of the modern Web architecture", ACM Trans. Inter. Tech. Vol. 2(2), pp. 115-150, DOI 10.1145/514183.514185, May 2002.
- [RFC4120]
Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005,
<<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4949]
Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007,
<<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC6347]
Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749]
Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012,
<<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7228]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014,
<<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230]
Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231]
Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014,
<<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7252]
Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC7744] Seitz, L., Ed., Gerdes, S., Ed., Selander, G., Mani, M., and S. Kumar, "Use Cases for Authentication and Authorization in Constrained Environments", RFC 7744, DOI 10.17487/RFC7744, January 2016, <<https://www.rfc-editor.org/info/rfc7744>>.
- [RFC8246] McManus, P., "HTTP Immutable Responses", RFC 8246, DOI 10.17487/RFC8246, September 2017, <<https://www.rfc-editor.org/info/rfc8246>>.

Acknowledgements

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

Authors' Addresses

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

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

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

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

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

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

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

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

Abstract

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

Status of This Memo

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
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
5.1. Communication Security Protocol	12
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

D.1.3.	aud (Audience) Claim	42
D.1.4.	exp (Expiration Time) Claim	42
D.1.5.	nbf (Not Before) Claim	42
D.1.6.	iat (Issued At) Claim	43
D.1.7.	cti (CWT ID) Claim	43
D.1.8.	cnf (Confirmation) Claim	43
D.1.9.	cks (COSE Key Structure) Claim	43
D.1.10.	aif (Authorization Information Format) Claim	43
D.2.	CBOR major types for Claims	43
D.3.	CBOR Web Token Example	44
	Authors' Addresses	44

1. Introduction

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

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

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

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

2. Terminology

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

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

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

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

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

3. Overview

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

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

Another building block is the lightweight web transfer protocol CoAP [RFC7252] for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP which further reduces overhead and message exchanges. When CoAP is used over UDP, transport layer security is provided by DTLS 1.2 [RFC6347] instead of TLS 1.2 [RFC5246].

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

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

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

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

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

3.1. OAuth 2.0

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

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

Access Tokens:

Access tokens are credentials used to access protected resources. An access token is a data structure representing authorization

permissions issued to the client. Access tokens are generated by the authorization server and consumed by the resource server. The access token is opaque to the client.

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

Proof of Possession Tokens:

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

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

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

Symmetric PoP key:

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

Asymmetric PoP key:

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

The access token is protected against modifications using a MAC or a digital signature of the AS. The choice of PoP key does not

necessarily imply a specific credential type for the integrity protection of the token. More information about PoP tokens can be found in [I-D.ietf-oauth-pop-architecture].

Scopes and Permissions:

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

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

Claims:

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

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

Introspection:

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

3.2. CoAP

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

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

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

3.3. Object Security

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

Object Security of CoAP (OSCOAP)

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

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

Object Security of Content (OSCON)

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

In this case, the message is not bound to the underlying application layer protocol and can therefore be used with HTTP, CoAP, Bluetooth Smart, etc. Whereas OSCOAP integrity protects specific CoAP message meta-data like request/response code, and

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

4. Protocol Interactions

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

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

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

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

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

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



Figure 1: Overview of the basic protocol flow

Requesting an Access Token (A):

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

Access Token Response (B):

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

Resource Request (C):

The client interacts with the RS to request access to the protected resource and provides the access token. The protocol to use between the client and the RS is not restricted to CoAP; HTTP, HTTP/2, Bluetooth Smart etc., are also possible candidates.

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

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

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

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

Token Introspection Request (D):

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

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

Token Introspection Response (E):

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

Protected Resource (F):

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

5. OAuth 2.0 Profiling

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

5.1. Communication Security Protocol

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

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

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

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

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

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

5.2. Authorization Information Resource at the Resource Server

A consequence of allowing the use of CoAP as web transfer protocol is that we cannot rely on HTTP specific mechanisms, such as transferring

information elements in HTTP headers since those are not necessarily gracefully mapped to CoAP. In case the access token is larger than 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 use 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.

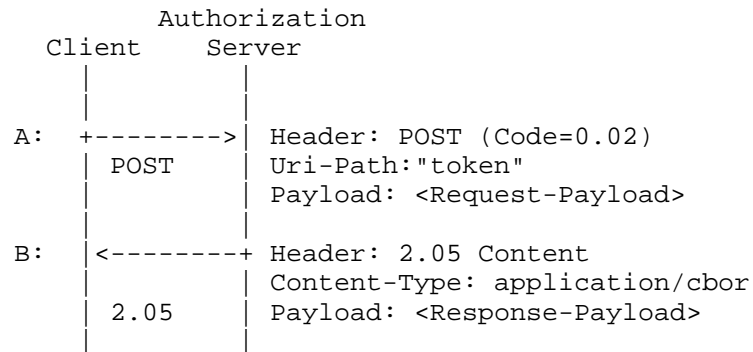


Figure 2: Token Request and Response Using Client Credentials.

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

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

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

Figure 3: Request and Response Payload Details.

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

```
{
  "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
  "kty" : "EC",
  "crv" : "P-256",
  "x" : b64'MKBCTNlckUSDiil1ySs3526iDZ8AiTo7Tu6KPAqv7D4',
```

```

    "y" : b64'4Et16SRW2YiLUrN5vfvVHuHp7x8PxltmWWlbbM4IFyM'
  }

```

Figure 4: Public Key of the RS.

```

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

```

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

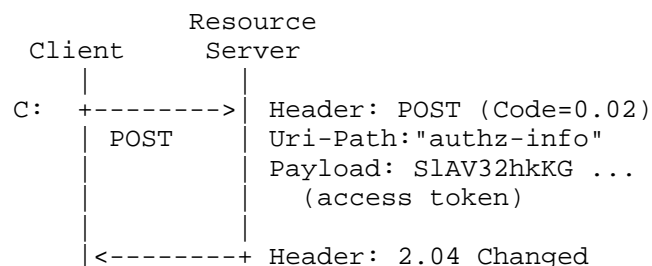
Messages C and F are shown in Figure 6 - Figure 7.

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

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

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

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



```

      | 2.04 |
      |     |

```

Figure 6: Access Token provisioning to RS

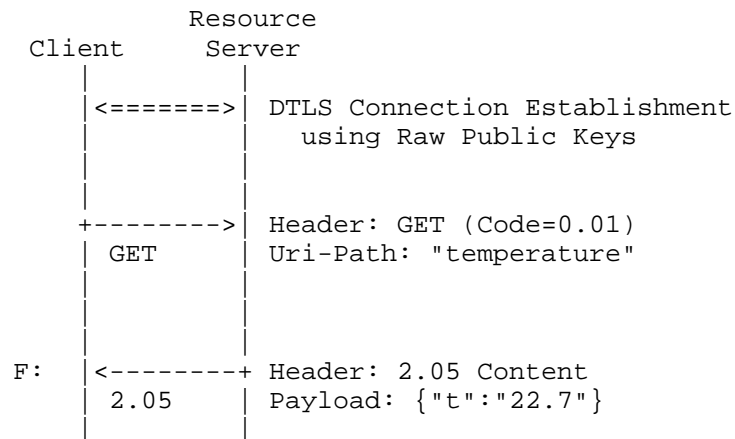


Figure 7: Resource Request and Response protected by DTLS.

6.2. Resource Server Offline

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

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

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

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

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

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

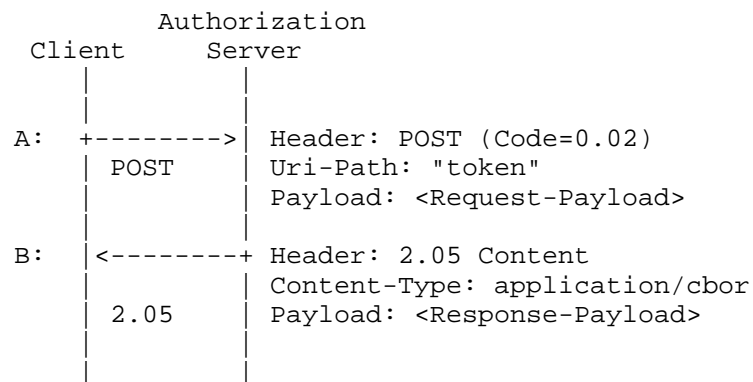


Figure 8: Token Request and Response

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_FzUA5XKMYoVHyzzff5oRJxl-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.

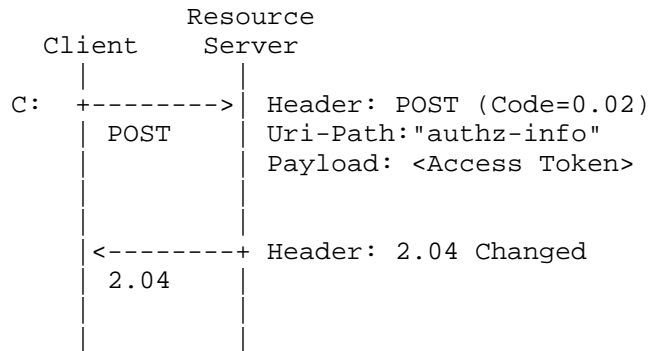


Figure 11: Access Token provisioning to RS

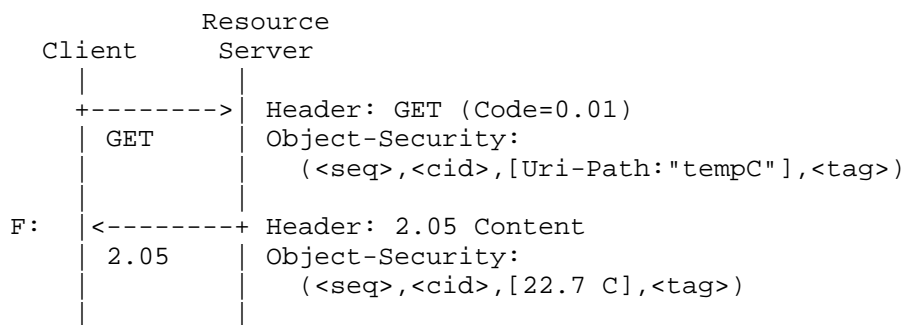


Figure 12: Resource request and response protected by OSCOAP

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

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

6.3. Token Introspection with an Offline Client

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

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

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

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

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

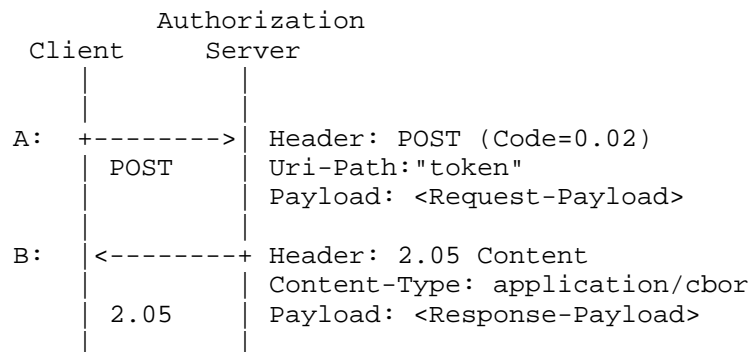


Figure 13: Token Request and Response using Client Credentials.

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

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

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

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

Figure 14: Request and Response Payload for C offline

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

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

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

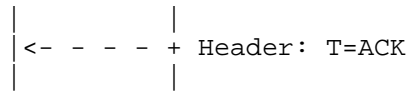


Figure 15: Access Token provisioning to RS

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.

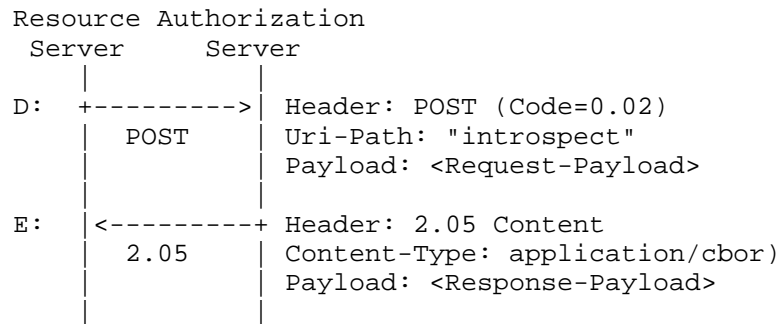


Figure 16: Token Introspection for C offline

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

```

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

Response-Payload:
{

```

```

    "active" : true,
    "aud" : "lockOfDoor4711",
    "scope" : "open, close",
    "iat" : 1311280970,
    "cnf" : {
      "ck" : b64'JDLUhtMjU2IiwiY3R5Ijoi ...'
    }
  }
}

```

Figure 17: Request and Response Payload for Introspection

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

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

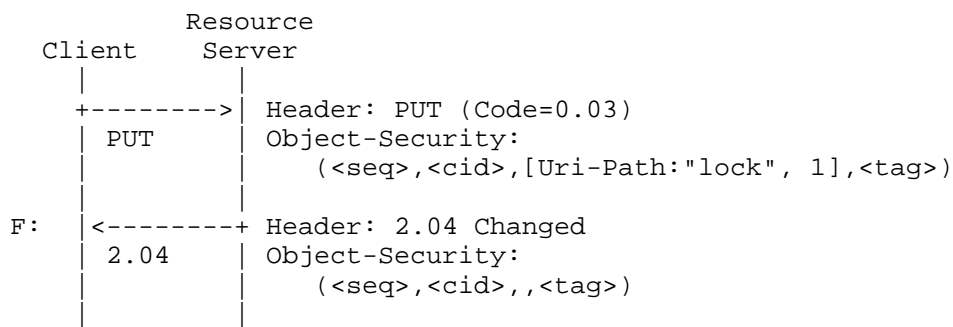


Figure 18: Resource request and response protected by OSCOAP

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

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

6.4. Always-On Connectivity

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

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

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

6.5. Token-less Authorization

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

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

1. The common case where authorization is performed by means of authentication fits into the same protocol framework. Authentication protocol and key is specified by client information, and access token is omitted.
2. Authentication, and thereby authorization, may even be implicit, i.e. anyone with access to the right key is authorized to access the protected resource.

In case 2., the RS does not need to receive any message from the client, and therefore enables offloading recurring resource request

and response processing to a third party, such as a Message Broker (MB) in a publish-subscribe setting.

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

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

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

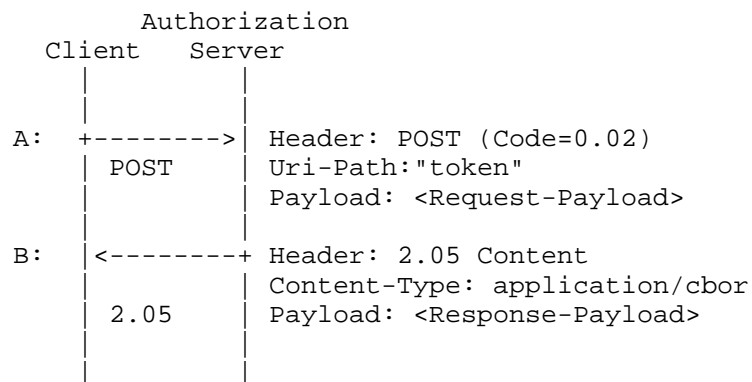


Figure 19: Token Request and Response

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

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



```

    "client_secret" : "qwerty"
  }

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

```

Figure 20: Request and Response Payload for RS severely constrained

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

```

key :
{
  "alg" : "AES_128_CTR_ECDSA",
  "kid" : b64'c29tZSBvdGhlciBrZXkgaWQ';
  "k" : b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE',
  "crv" : "P-256",
  "x" : b64'MKBCTNlckUSDiillySs3526iDZ8AiTo7Tu6KPAqv7D4',
  "y" : b64'4Et16SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
}

```

Figure 21: The 'key' Parameter

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

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

Message exchanges C and F are shown in Figure 22.

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

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

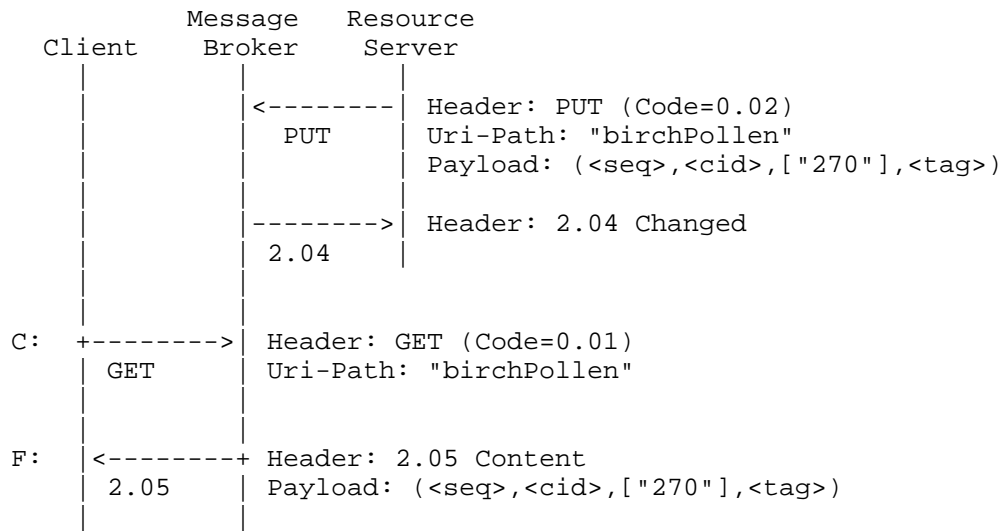


Figure 22: Sensor measurement protected by COSE

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

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

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

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

6.6. Securing Group Communication

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

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

7. Security Considerations

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

8. IANA Considerations

TBD

9. Acknowledgments

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

10. References

10.1. Normative References

- [I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-02 (work in progress), March 2015.
- [I-D.ietf-cose-msg]
Schaad, J. and B. Campbell, "CBOR Encoded Message Syntax", draft-ietf-cose-msg-05 (work in progress), September 2015.
- [I-D.ietf-oauth-introspection]
Richer, J., "OAuth 2.0 Token Introspection", draft-ietf-oauth-introspection-09 (work in progress), May 2015.
- [I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-02 (work in progress), July 2015.
- [I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-01 (work in progress), March 2015.
- [I-D.selander-ace-object-security]
Selander, G., Mattsson, J., and L. Seitz, "March 9, 2015", draft-selander-ace-object-security-01 (work in progress), March 2015.
- [I-D.wahlstroem-ace-oauth-introspection]
Wahlstroem, E., "OAuth 2.0 Introspection over the Constrained Application Protocol (CoAP)", draft-wahlstroem-ace-oauth-introspection-01 (work in progress), March 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

10.2. Informative References

- [I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-00 (work in progress), August 2015.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.
- [I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., and H. Tschofenig, "Multicast Security for the Lighting Domain", draft-somaraju-ace-multicast-00 (work in progress), July 2015.
- [RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, October 2006.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.
- [RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <<http://www.rfc-editor.org/info/rfc6819>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.
- [RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.

Appendix A. Design Justification

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

Common IoT constraints are:

Low Power Radio:

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

Low CPU Speed:

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

Small Amount of Memory:

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

User Interface Limitations:

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

Communication Constraints:

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

The communication interactions this framework builds upon (as shown graphically in Figure 1) may be accomplished using a variety of different protocols, and not all parts of the message flow are

used in all applications due to the communication constraints. While we envision deployments to make use of CoAP we explicitly want to support HTTP, HTTP/2 or specific protocols, such as Bluetooth Smart communication, which does not necessarily use IP. The latter raises the need for application layer security over the various interfaces.

Appendix B. Optimizations

This section sketches some potential optimizations to the presented solution.

Access token in DTLS handshake

In the case of CSP=DTLS/TLS, the access token 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.

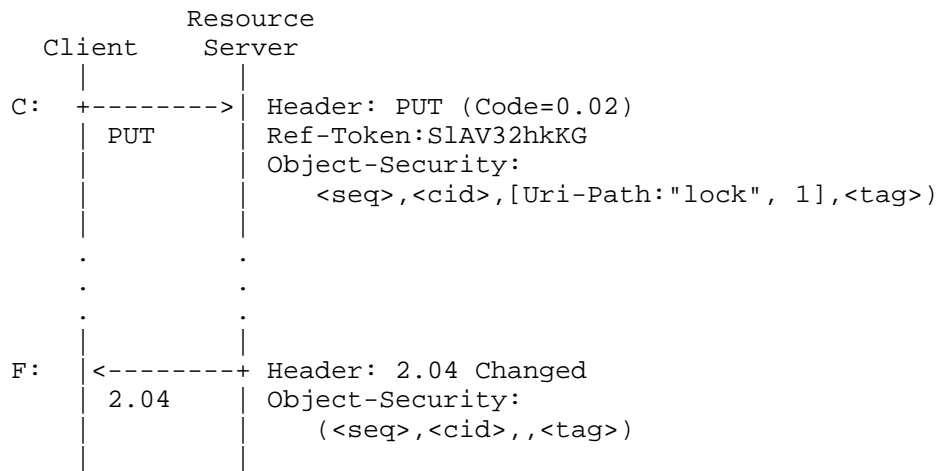


Figure 23: Reference Token in CoAP Option

Appendix C. CoAP and CBOR profiles for OAuth 2.0

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

C.1. Profile for Token resource

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

C.1.1. Token Request

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

grant_type:

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

client_id:

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

client_secret:

OPTIONAL. The client secret.

scope:

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

aud:

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

alg:

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

key:

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

The parameters defined above use the following CBOR major types.

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

Figure 24: CBOR mappings used in token requests

C.1.2. Token Response

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

access_token:

REQUIRED. The access token issued by the authorization server.

token_type:

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

key:

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

csp:

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

scope:

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

alg:

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

The parameters defined above use the following CBOR major types.

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

\-----+-----+-----/

Figure 25: CBOR mappings used in token responses

C.2. CoAP Profile for OAuth Introspection

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

C.2.1. Introspection Request

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

token:

REQUIRED. The string value of the token.

resource_id:

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

client_id:

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

client_secret:

OPTIONAL. The client secret.

The parameters defined above use the following CBOR major types:

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

Figure 26: CBOR Mappings to Token Introspection Request Parameters.

C.2.2. Introspection Response

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

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

Introspection responses in CBOR use the following mappings:

active:

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

scope:

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

client_id:

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

username:

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

token_type:

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

exp:

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

iat:

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

nbf:

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

sub:

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

aud:

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

iss:

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

cti:

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

The parameters defined above use the following CBOR major types:

/-----+-----+-----\		
Value	Major Type	Key
-----+-----+-----		

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

Figure 27: CBOR Mappings to Token Introspection Response Parameters.

Appendix D. CBOR Web Token (CWT)

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

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

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

D.1. Claim Names

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

D.1.1. iss (Issuer) Claim

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

D.1.2. sub (Subject) Claim

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

D.1.3. aud (Audience) Claim

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

D.1.4. exp (Expiration Time) Claim

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

D.1.5. nbf (Not Before) Claim

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

D.1.6. iat (Issued At) Claim

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

D.1.7. cti (CWT ID) Claim

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

D.1.8. cnf (Confirmation) Claim

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

D.1.9. cks (COSE Key Structure) Claim

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

D.1.10. aif (Authorization Information Format) Claim

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

D.2. CBOR major types for Claims

/-----+-----+-----\			
Value	Major Type	Key	
0	0	iss	
1	0	sub	
2	0	aud	
3	0	nonce	
4	0	exp	

5	0	iat	
6	4	cnf	
7	0	ck	
8	4	aif	
\-----+-----+-----+-----/			

Figure 28: CBOR Mappings used in CWT Access Tokens.

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

D.3. CBOR Web Token Example

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

```
{
  "iss" : "coap://as.example.com",
  "aud" : "coap://light.example.com",
  "exp" : 1444064944,
  "iat" : 1443944944,
  "aif" : [ ["/s/light", 1], ["/a/led", 5], ["/dltls", 2] ],
  "cnf" : {
    "jwk" : b64'JDLUHTMjU2IiwiY3R5Ijoi ...'
  }
}
```

Figure 29: CWT Example in the CBOR Diagnostic Notation.

Authors' Addresses

Ludwig Seitz
 SICS
 Scheelevaegen 17
 Lund 223 70
 SWEDEN

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
SWEDEN

Email: goran.selander@ericsson.com

Erik Wahlstroem
Nexus Technology
Telefonvagen 26
Hagersten 126 26
Sweden

Email: erik.wahlstrom@nexusgroup.com

Samuel Erdtman
Nexus Technology
Telefonvagen 26
Hagersten 126 26
Sweden

Email: samuel.erdman@nexusgroup.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com