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

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI
G. Selander
Ericsson
L. Seitz
RISE SICS

March 13, 2017

Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE) draft-gerdes-ace-dtls-authorize-01

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

This specification defines a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use CoAP [RFC7252] over DTLS [RFC6347] to communicate. The client uses an access token, bound to a key (the proof-of-possession key) to authorize its access to the resource server. DTLS provides communication security, proof of possession, and server authentication. Optionally the client and the resource server may also use CoAP over DTLS to communicate with the authorization server. This specification supports the DTLS PSK handshake [RFC4279] and the DTLS handshake with Raw Public Keys (RPK) [RFC7250].

The DTLS PSK handshake [RFC4279] provides the proof-of-possession for the key tied to the access token. Furthermore the psk_identity parameter in the DTLS PSK handshake is used to transfer the access token from the client to the resource server.

The DTLS RPK handshake [RFC7250] requires client authentication to provide proof-of-possession for the key tied to the access token. Here the access token needs to be transferred to the resource server before the handshake is initiated, as described in section 8.1 of draft-ietf-ace-oauth-authz. [1]

Note: While the scope of this draft is on client and resource server

communicating using CoAP over DTLS, it is expected that it applies also to CoAP over TLS, possibly with minor modifications. However, that is out of scope for this version of the draft.

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 expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz].

2. Protocol Overview

The CoAP-DTLS profile for ACE specifies the transfer of authentication and, if necessary, authorization information between C and RS during setup of a DTLS session for CoAP messaging. It also specifies how a Client can use CoAP over DTLS to retrieve an Access Token from AS for a protected resource hosted on RS.

This profile requires a Client (C) to retrieve an Access Token for the resource(s) it wants to access on a Resource Server (RS) as specified in [I-D.ietf-ace-oauth-authz]. Figure 1 shows the typical message flow in this scenario (messages in square brackets are optional):

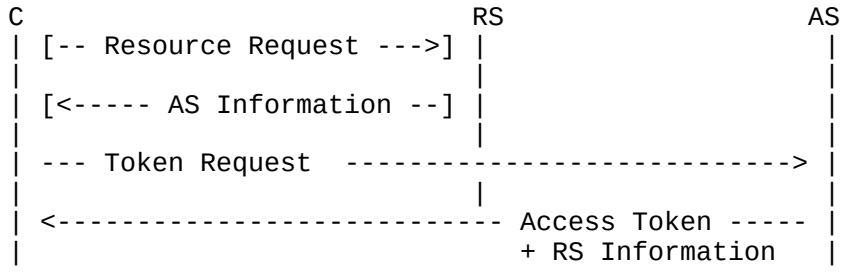


Figure 1: Retrieving an Access Token

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

Once C knows AS's address, it can send an Access Token request to the /token endpoint at the AS as specified in [I-D.ietf-ace-oauth-authz]. If C wants to use the CoAP RawPublicKey mode as described in Section 9 of RFC 7252 [2] it MUST provide a key or key identifier within a "cnf" object in the token request. If AS decides that the request is to be authorized it generates an access token response for C containing a "profile" parameter with the value "coap_dtls" to indicate that this profile MUST be used for communication between C and RS. Is also adds a "cnf" parameter with additional data for the establishment of a secure DTLS channel between C and RS. The semantics of the 'cnf' parameter depend on the type of key used between C and RS, see Section 3 and Section 4.

The Access Token returned by AS then can be used by C to establish a new DTLS session with RS. When C intends to use asymmetric cryptography in the DTLS handshake with RS, C MUST upload the Access Token to the "/authz-info" resource on RS before starting the DTLS handshake, as described in section 8.1 of draft-ietf-ace-oauth-authz [3]. If only symmetric cryptography is used between C and RS, the Access Token MAY instead be transferred in the DTLS ClientKeyExchange message (see Section 4.1).

Figure 2 depicts the common protocol flow for the DTLS profile after C has retrieved the Access Token from AS.

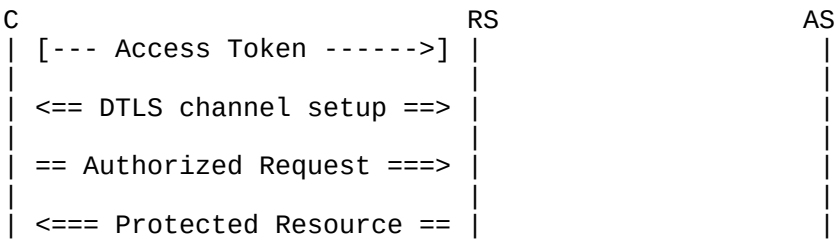


Figure 2: Protocol overview

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

Depending on the desired CoAP security mode, the Client-to-AS request, AS-to-Client response and DTLS session establishment carry slightly different information. Section 3 addresses the use of raw public keys while Section 4 defines how pre-shared keys are used in this profile.

2.1. Unauthorized Resource Request Message

The optional Unauthorized Resource Request message is a request for a resource hosted by RS for which no proper authorization is granted. RS MUST treat any CoAP request for a resource other than "/authz-info" as Unauthorized Resource Request message when any of the following holds:

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

Note: These conditions ensure that RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The resource "/authz-info" is publicly accessible to be able to upload new access tokens to RS (cf. [I-D.ietf-ace-oauth-authz]).

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Resource Server SHOULD provide proper AS Information to enable the Client to request an access token from RS's Authorization Server as described in Section 2.2.

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

2.2. AS Information

The AS Information is sent by RS as a response to an Unauthorized Resource Request message (see Section 2.1) to point the sender of the Unauthorized Resource Request message to RS's AS. The AS information is a set of attributes containing an absolute URI (see Section 4.3 of [RFC3986]) that specifies the AS in charge of RS.

TBD: We might not want to add more parameters in the AS information because
this would not only reveal too much information about RS's capabilities to unauthorized peers but also be of little value as C cannot really trust that information anyway.

The message MAY also contain a nonce generated by RS to ensure freshness in case that the RS and AS do not have synchronized clocks.

Figure 3 shows an example for an AS Information message payload using CBOR [RFC7049] diagnostic notation.

```
4.01 Unauthorized
Content-Format: application/ace+cbor
{AS: "coaps://as.example.com/token",
 nonce: h'e0a156bb3f'}
```

Figure 3: AS Information payload example

In this example, the attribute AS points the receiver of this message to the URI "coaps://as.example.com/token" to request access permissions. The originator of the AS Information payload (i.e., RS) uses a local clock that is loosely synchronized with a time scale common between RS and AS (e.g., wall clock time). Therefore, it has included a parameter "nonce" for replay attack prevention (c.f. Section 5.2).

Note: There is an ongoing discussion how freshness of access tokens can be achieved in constrained environments. This specification for now assumes that RS and AS do not have a common understanding of time that allows RS to achieve its security objectives without explicitly adding a nonce.

The examples in this document are written in CBOR diagnostic notation to improve readability. Figure 4 illustrates the binary encoding of the message payload shown in Figure 3.

```
a2                                  # map(2)
    00                              # unsigned(0) (=AS)
    78 1c                           # text(28)
       636f6170733a2f2f61732e657861
       6d706c652e636f6d2f746f6b656e  # "coaps://as.example.com/token"
    05                              # unsigned(5) (=nonce)
    45                              # bytes(5)
       e0a156bb3f
```

Figure 4: AS Information example encoded in CBOR

2.3. Resource Access

Once a DTLS channel has been established as described in Section 3 and Section 4, respectively, C is authorized to access resources covered by the Access Token it has uploaded to the "/authz-info" resource hosted by RS.

On the server side (i.e., RS), successful establishment of the DTLS channel binds C to the access token, functioning as a proof-of-possession associated key. Any request that RS receives on this channel MUST be checked against these authorization rules that are associated with the identity of C. Incoming CoAP requests that are not authorized with respect to any Access Token that is associated with C MUST be rejected by RS with 4.01 response as described in Section 2.1.

Note: The identity of C is determined by the authentication process

during the DTLS handshake. In the asymmetric case, the public key will define C's identity, while in the PSK case, C's identity is defined by the session key generated by AS for this communication.

RS SHOULD treat an incoming CoAP request as authorized if the following holds:

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

2. The authorization information tied to the sending peer is valid.

3. The request is destined for RS.

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.

Incoming CoAP requests received on a secure DTLS channel 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.

C cannot always know a priori if a Authorized Resource Request will succeed. If C repeatedly gets AS Information messages (cf. Section 2.2) as response to its requests, it SHOULD request a new Access Token from AS in order to continue communication with RS.

2.4. Dynamic Update of Authorization Information

The Client can update the authorization information stored at RS at any time. To do so, the Client requests from AS a new Access Token for the intended action on the respective resource and uploads this Access Token to the "/authz-info" resource on RS.

Figure 5 depicts the message flow where C requests a new Access Token after a security association between C and RS has been established using this protocol.

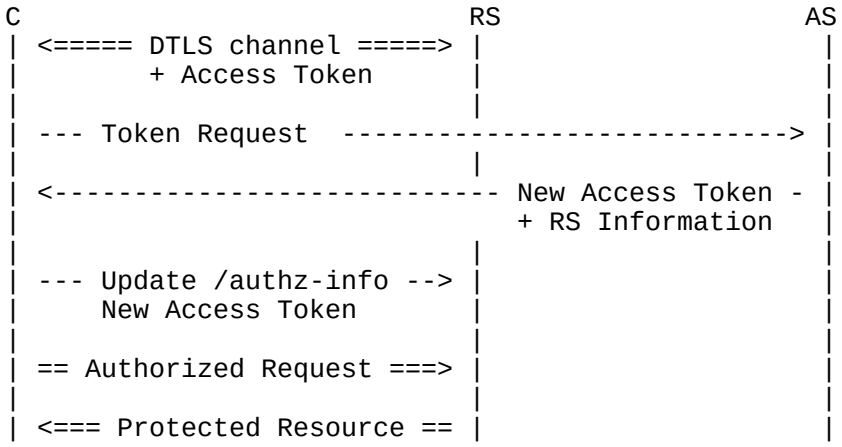


Figure 5: Overview of Dynamic Update Operation

3. RawPublicKey Mode

To retrieve an access token for the resource that C wants to access, C requests an Access Token from AS. C MUST add a "cnf" object carrying either its raw public key or a unique identifier for a public key that it has previously made known to AS.

An example Access Token request from C to RS is depicted in Figure 6.

```
POST coaps://as.example.com/token
Content-Format: application/cbor
{
  grant_type:    client_credentials,
  aud:           "tempSensor4711",
  cnf: {
    COSE_Key: {
      kty: EC2,
      crv: P-256,
      x:   h'TODOX',
      y:   h'TODOY'
    }
  }
}
```

Figure 6: Access Token Request Example for RPK Mode

The example shows an Access Token request for the resource identified by the audience string "tempSensor4711" on the AS using a raw public key.

When AS authorizes a request, it will return an Access Token and a "cnf" object in the AS-to-Client response. Before C initiates the DTLS handshake with RS, it MUST send a "POST" request containing the new Access Token to the "/authz-info" resource hosted by RS. If this operation yields a positive response, C SHOULD proceed to establish a new DTLS channel with RS. To use raw public key mode, C MUST pass the same public key that was used for constructing the Access Token with the SubjectPublicKeyInfo structure in the DTLS handshake as specified in [RFC7250].

Note: According to [RFC7252], CoAP implementations MUST support the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251] and the NIST P-256 curve. C is therefore expected to offer at least this ciphersuite to RS.

The Access Token is constructed by AS such that RS can associate the Access Token with the Client’s public key. If CBOR web tokens [I-D.ietf-ace-cbor-web-token] are used as recommended in [I-D.ietf-ace-oauth-authz], the AS MUST include a "COSE_Key" object in the "cnf" claim of the Access Token. This "COSE_Key" object MAY contain a reference to a key for C that is already known by RS (e.g., from previous communication). If the AS has no certain knowledge that the Client’s key is already known to RS, the Client’s public key MUST be included in the Access Token’s "cnf" parameter.

4. PreSharedKey Mode

To retrieve an access token for the resource that C wants to access, C MAY include a "cnf" object carrying an identifier for a symmetric key in its Access Token request to AS. This identifier can be used by AS to determine the session key to construct the proof-of-possession token and therefore MUST specify a symmetric key that was previously generated by AS as a session key for the communication between C and RS.

Depending on the requested token type and algorithm in the Access Token request, AS adds RS Information to the response that provides C with sufficient information to setup a DTLS channel with RS. For symmetric proof-of-possession keys (c.f. [I-D.ietf-ace-oauth-authz]), C must ensure that the Access Token request is sent over a secure channel that guarantees authentication, message integrity and confidentiality. This could be, e.g., a DTLS channel (for "coaps") or an OSCOAP [I-D.ietf-core-object-security] exchange (for "coap").

When AS authorizes C it returns an AS-to-Client response with the profile parameter set to "coap_dtls" and a "cnf" parameter carrying a

"COSE_Key" object that contains the symmetric session key to be used between C and RS as illustrated in Figure 7.

```
2.01 Created
Content-Format: application/cbor
Location-Path: /token/asdjbaskd
Max-Age: 86400
{
   access_token: b64'SlAV32hkKG ...
   (remainder of CWT omitted for brevity;
   token_type:   pop,
   alg:          HS256,
   expires_in:   86400,
   profile:      coap_dtls,
   cnf: {
     COSE_Key: {
       kty: symmetric,
       k: h'73657373696f6e6b6579'
     }
   }
}
```

Figure 7: Example Access Token response

In this example, AS returns a 2.01 response containing a new Access Token. The information is transferred as a CBOR data structure as specified in [I-D.ietf-ace-oauth-authz]. The Max-Age option tells the receiving Client how long this token will be valid.

A response that declines any operation on the requested resource is constructed according to Section 5.2 of RFC 6749 [4], (cf. Section 6.3 of [I-D.ietf-ace-oauth-authz]).

```
4.00 Bad Request
Content-Format: application/cbor
{
  error: invalid_request
}
```

Figure 8: Example Access Token response with reject

4.1. DTLS Channel Setup Between C and RS

When C receives an Access Token from AS, it checks if the payload contains an "access_token" parameter and a "cnf" parameter. With this information C can initiate establishment of a new DTLS channel with RS. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279] using the key

conveyed in the "cnf" parameter of the AS response as PSK when constructing the premaster secret.

In PreSharedKey mode, the knowledge of the session key by C and RS is used for mutual authentication between both peers. Therefore, RS must be able to determine the session key from the Access Token. Following the general ACE authorization framework, C can upload the Access Token to RS's "/authz-info" resource before starting the DTLS handshake. Alternatively, C MAY provide the most recent base64-encoded Access Token in the "psk_identity" field of the ClientKeyExchange message.

If RS receives a ClientKeyExchange message that contains a "psk_identity" with a length greater zero, it MUST base64-decode its contents and check if the "psk_identity" field contains a key identifier or Access Token according to the following CDDL specification:

```
psk_identity = {
  kid => bstr / access_token => bstr
}
```

The identifiers for the map keys "kid" and "access_token" are used with the same meaning as in COSE [I-D.ietf-cose-msg] and the ACE framework [I-D.ietf-ace-oauth-authz] respectively. The identifier "kid" thus has the value 4 (see [I-D.ietf-cose-msg]), and the identifier "access_token" has the value 19, respectively (see [I-D.ietf-ace-oauth-authz]).

If the "psk_identity" field contains a key identifier, the receiver MUST check if it has one or more Access Tokens that are associated with the specified key. If no valid Access Token is available for this key, the DTLS session setup is terminated with an "illegal_parameter" DTLS alert message.

If instead the "psk_identity" field contains an Access Token, it must processed in the same way as an Access Token that has been uploaded to its "/authz-info" resource. In this case, RS continues processing the ClientKeyExchange message if the contents of the "psk_identity" contained a valid Access Token. Otherwise, the DTLS session setup is terminated with an "illegal_parameter" DTLS alert message.

Note1: As RS cannot provide C with a meaningful PSK identity hint in

response to C's ClientHello message, RS 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 RS.

This specification assumes that the Access Token is a PoP token as described in [I-D.ietf-ace-oauth-authz] unless specifically stated otherwise. Therefore, the Access Token is bound to a symmetric PoP key that is used as session key between C and RS.

While C can retrieve the session key from the contents of the "cnf" parameter in the AS-to-Client response, RS uses the information contained in the "cnf" claim of the Access Token to determine the actual session key when no explicit "kid" was provided in the "psk_identity" field. Usually, this is done by including a "COSE_Key" object carrying either a key that has been encrypted with a shared secret between AS and RS, or a key identifier that can be used by RS to lookup the session key.

Instead of the "COSE_Key" object, AS MAY include a "COSE_Encrypt" structure to enable RS to calculate the session key from the Access Token. The "COSE_Encrypt" structure MUST use the _Direct Key with KDF_ method as described in Section 12.1.2 of draft-ietf-cose-msg [5]. The AS MUST include a Context information structure carrying a PartyU "nonce" parameter carrying the nonce that has been used by AS to construct the session key.

This specification mandates that at least the key derivation algorithm "HKDF SHA-256" as defined in [I-D.ietf-cose-msg] MUST be supported. This key derivation function is the default when no "alg" field is included in the "COSE_Encrypt" structure for RS.

4.2. Updating Authorization Information

Usually, the authorization information that RS keeps for C is updated by uploading a new Access Token as described in Section 2.4.

If the security association with RS still exists and RS has indicated support for session renegotiation according to [RFC5746], the new Access Token MAY be used to renegotiate the existing DTLS session. In this case, the Access Token is used as "psk_identity" as defined in Section 4.1. The Client MAY also perform a new DTLS handshake according to Section 4.1 that replaces the existing DTLS session.

After successful completion of the DTLS handshake RS updates the existing authorization information for C according to the new Access Token.

5. Security Considerations

TODO

5.1. Unprotected AS Information

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the AS information contained in an unprotected response from RS to an unauthorized request (c.f. Section 2.2) is authentic. It is therefore advisable to provide C with a (possibly hard-coded) list of trustworthy authorization servers. AS information responses referring to a URI not listed there would be ignored.

5.2. Use of Nonces for Replay Protection

RS may add a nonce to the AS Information message sent as a response to an unauthorized request to ensure freshness of an Access Token subsequently presented to RS. While a timestamp of some granularity would be sufficient to protect against replay attacks, using randomized nonce is preferred to prevent disclosure of information about RS's internal clock characteristics.

5.3. Privacy

An unprotected response to an unauthorized request (c.f. Section 2.2) may disclose information about RS and/or its existing relationship with C. It is advisable to include as little information as possible in an unencrypted response. When a DTLS session between C and RS already exists, more detailed information may be included with an error response to provide C with sufficient information to react on that particular error.

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-05 (work in progress), February 2017.

[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>.

[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>.

[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>.

7.2. Informative References

[I-D.ietf-ace-cbor-web-token] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-03 (work in progress), March 2017.

[I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-09 (work in progress), October 2016.

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

[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>.

[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>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <http://www.rfc-editor.org/info/rfc7250>.

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <http://www.rfc-editor.org/info/rfc7251>.

7.3. URIs

[1] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-03#section-8.1

[2] https://tools.ietf.org/html/rfc7252#section-9

[3] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-03#section-8.1

[4] https://tools.ietf.org/html/rfc6749#section-5.2

[5] https://tools.ietf.org/html/draft-ietf-cose-msg-23#section-12.1.2

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

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: September 7, 2017

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

March 06, 2017

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

Abstract

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 7, 2017.

Copyright Notice

Table of Contents

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.)

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

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

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

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

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

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:

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

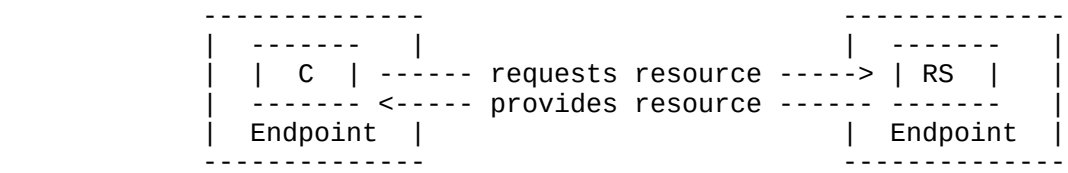


Figure 1: Constrained Level

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

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

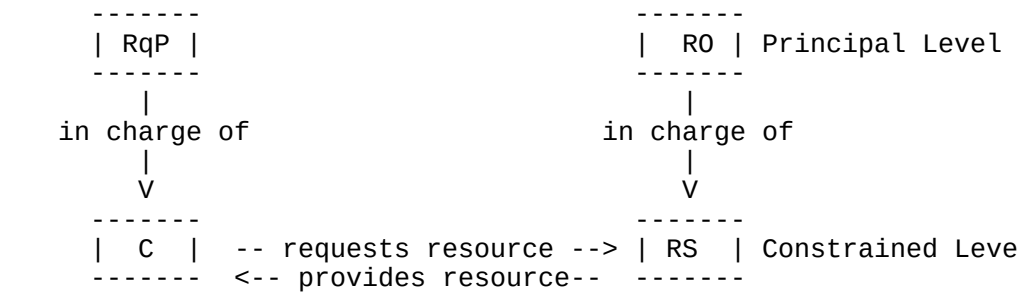


Figure 2: Constrained Level and Principal Level

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

Moreover, constrained endpoints may need support with tasks requiring heavy processing, large memory or storage, or interfacing to humans, such as management of security policies defined by a principal. The

principal, in turn, requires some agent maintaining the policies governing how its endpoints will interact.

For these reasons, another level of nodes is introduced in the architecture, the less-constrained level (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 principals. Many of these entities, often acting for different principals, can be combined into a single server implementation; this of course requires proper segregation of the control information provided by each principal.)

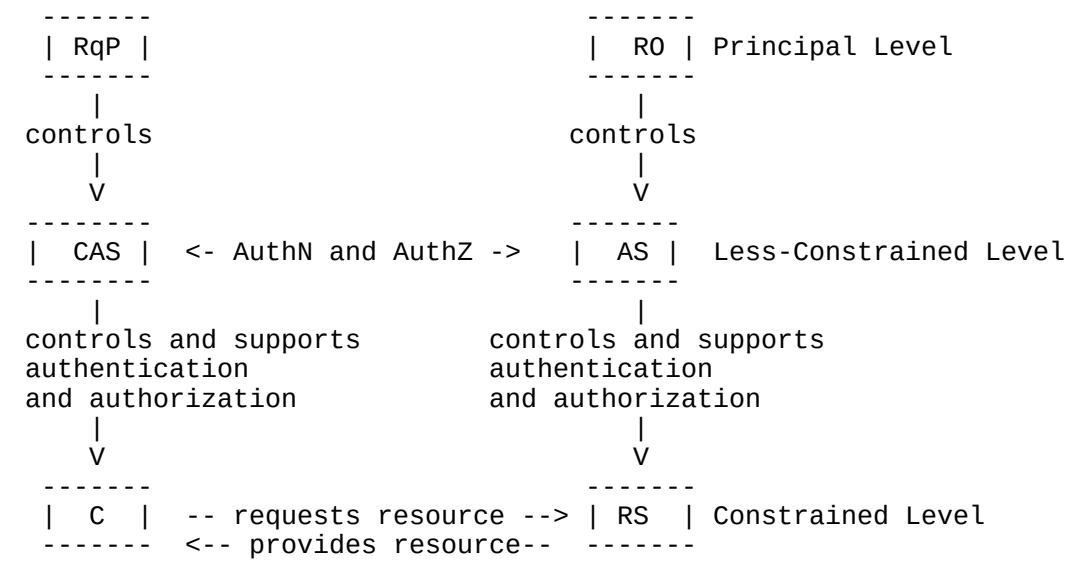


Figure 3: Overall architecture

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:

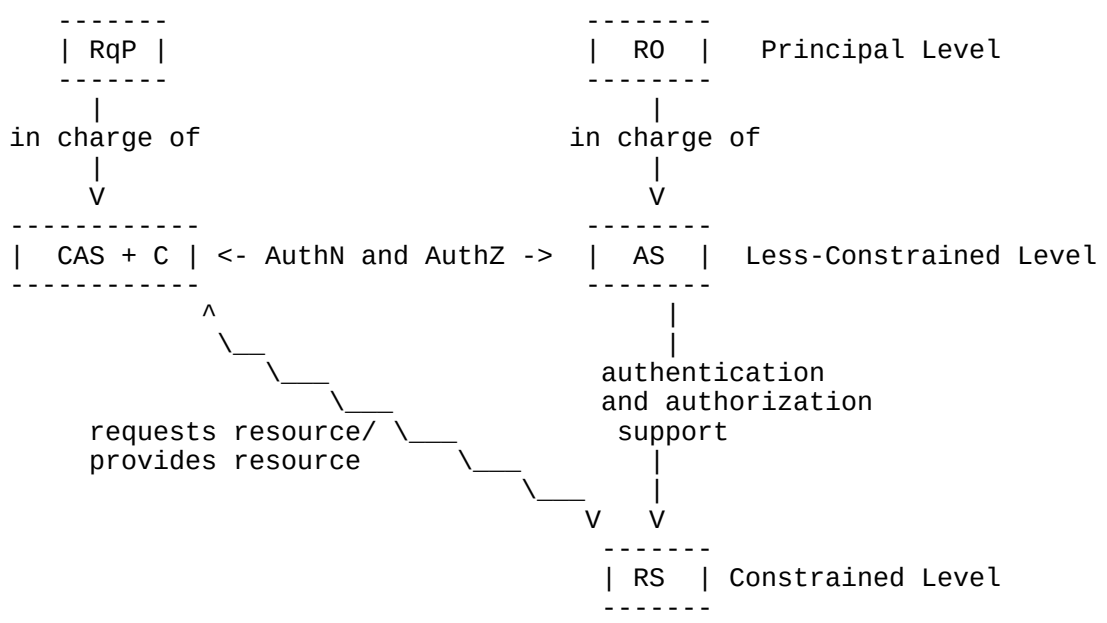


Figure 4: Combined C and CAS

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

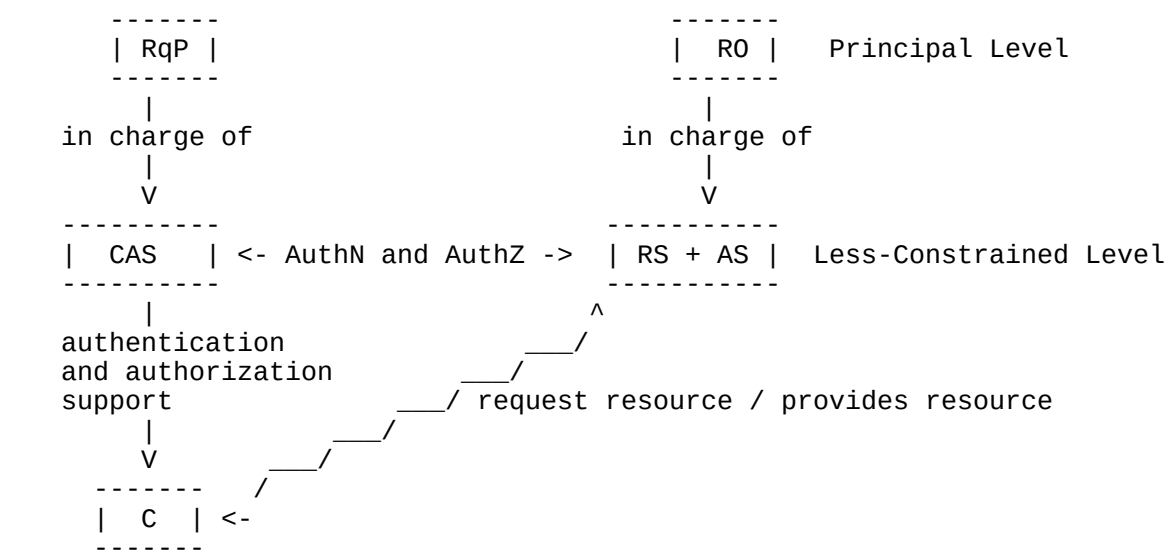


Figure 5: Combined AS and RS

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

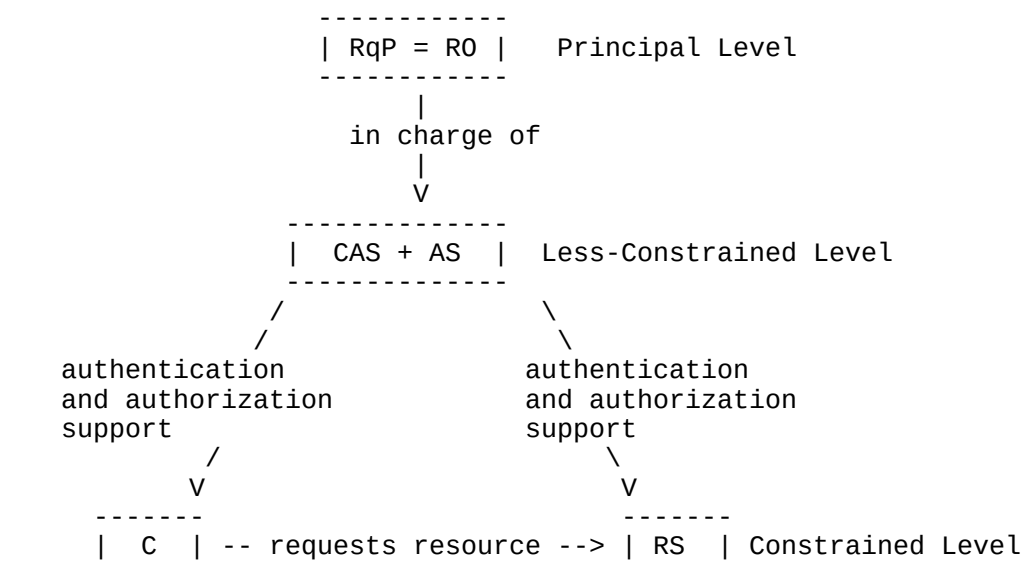


Figure 6: CAS combined with AS

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, reflecting the intrinsic asymmetry with C initiating the request for access to a resource, and RS acting on a received request, and C finally acting on the received response.

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

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

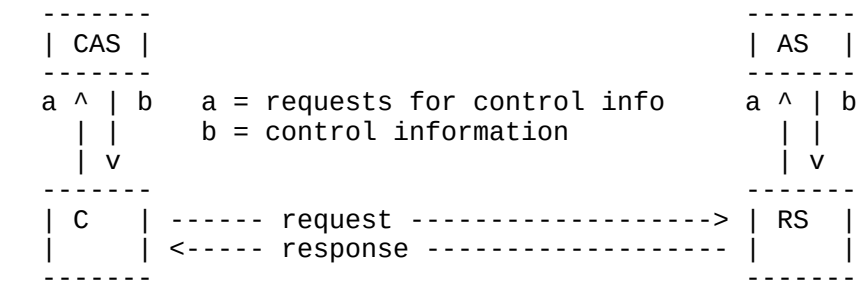


Figure 7: Information flows that need to be protected

- We assume that the necessary keys/credentials for protecting the control information between the potentially constrained nodes and their associated less-constrained nodes are pre-established, for example as part of the commissioning procedure.

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

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:

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

- 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.

- 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, allowing only selected operations by selected entities limits the burden on system resources, thus helping to achieve availability. Misconfigured or wrongly designed authorization solutions can result in availability breaches (denial of service): 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:

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

- To comply with requirements of transparency, a gateway may be allowed to read, verify (authenticity) but not modify (integrity) a resource representation which therefore also is end-to-end

integrity protected from the server towards a client behind the gateway.

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

4. Authentication and Authorization

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

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

The security objectives the authorization mechanism aims at can only be achieved if the authentication and the authorization mechanism work together correctly. We speak of authenticated authorization to refer to the required synthesis of 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 other scenarios, there is often less need to uniquely identify an individual device: For a principal, the fact that a device belongs to a certain company or that it has a specific type (such as a light bulb) or location may be more important than that it has a unique identifier.

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

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

Authorization granularity	Authorization is contingent on:
device	authentication of specific device
owner	(authenticated) authorization by owner
flat	(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. 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, flat 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; a constrained client may not have that available all the time but still needs to implement the wishes of the 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. Principals may decide to omit authentication (unrestricted authorization), or use flat authorization (just employing an authentication mechanism). However, as indicated in Section 3, the security objectives of both sides must be considered, which can often only be achieved when the other side can be relied on to perform some security service.

5. Actors and their Tasks

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

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

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

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

5.1. Constrained Level Actors

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

C performs the following tasks:

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

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

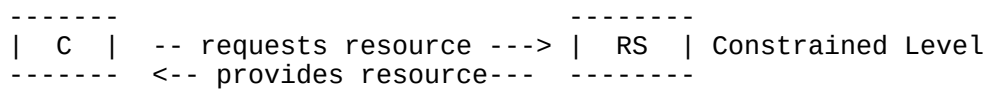


Figure 8: Constrained Level Actors

5.2. Principal Level Actors

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

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

RqP must fulfill the following task:

- Configure for C authorization information for sources for R.

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

RO must fulfill the following task:

- Configure for RS authorization information for accessing R.

(See Figure 2.)

5.3. Less-Constrained Level Actors

Constrained level actors can only fulfill a limited number of tasks and may not have network connectivity all the time. To relieve them from having to manage keys for numerous endpoints and conducting computationally intensive tasks, another complexity level for actors is introduced. An actor on the less-constrained level belongs to the same security domain as its respective constrained level actor. They also have the same principal.

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

CAS performs the following tasks:

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

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

AS performs the following tasks:

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

6. Kinds of Protocols

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

6.1. Constrained Level Protocols

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

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

On the constrained level, protocols need to address the specific requirements of constrained environments. Examples for protocols that consider these requirements is the transfer protocol CoAP (Constrained Application Protocol) [RFC7252] and the Datagram

Transport Layer Security Protocol (DTLS) [RFC6347] which can be used for channel security.

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

6.1.1. Cross Level Support Protocols

Protocols which operate between a constrained device on one side and the corresponding less-constrained device on the other are considered to be (cross level) support protocols. Protocols used between C and CAS or RS and AS are therefore support protocols.

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

6.2. Less-Constrained Level Protocols

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

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

7. Elements of a Solution

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

7.1. Authorization

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

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

- The transferred authorization information needs to follow a defined format and encoding, which must be efficient for

constrained devices, considering size of authorization information and parser complexity.

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

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

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

7.2. Authentication

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

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

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

- In some use cases RS needs to authenticate some property of C, in order to map it to the relevant authorization information. In other applications, authentication and authorization of C may be implicit, for example by encrypting the resource representation the RS only providing access to those who possess the key to decrypt.

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

- CAS and AS need to authenticate their communication partner (C or RS), in order to ensure it serves the correct device.

7.3. Communication Security

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

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

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

- Hybrid solutions using both session-based and object security are also possible. An example of a hybrid is where authorization information and cryptographic keys are provided by AS in the format of secure data objects, but where the resource access is protected by session-based security.

7.4. Cryptographic Keys

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

Symmetric vs Asymmetric Keys
: We need keys both for protection of resource access and for protection of transport of authentication and authorization information. Do we want to support solutions based on asymmetric keys or symmetric keys in both cases? There are classes of devices that can easily perform symmetric cryptography, but consume considerably more time/battery for asymmetric operations. On the other hand asymmetric cryptography has benefits such as in terms of deployment.

Key Establishment
: How are the corresponding cryptographic keys established? Considering Section 7.1 there must be a mapping between these keys and the authorization information, at least in the sense that AS must be able to specify a unique client identifier which RS can

verify (using an associated key). One of the use cases of [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.

8.1. Architecture

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

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

8.2. Constrained Devices

- C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover:
  - unable to manage complex authorization policies
  - unable to manage a large number of secure connections
  - without user interface
  - without constant network connectivity
  - unable to precisely measure time
  - required to save on wireless communication due to high power consumption
- CAS and AS are not assumed to be constrained devices.
- All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.
  - We assume the use of a standardized symmetric key algorithm, such as AES.
  - Except for the most constrained devices we assume the use of a standardized cryptographic hash function such as SHA-256 (which can be used with the HMAC construction for integrity protection).
- Public key cryptography requires additional resources (such as RAM, ROM, power, specialized hardware).
- A DTLS handshake involves significant computation, communication, and memory overheads in the context of constrained devices.
  - The RAM requirements of DTLS handshakes with public key cryptography are prohibitive for certain constrained devices.
  - Certificate-based DTLS handshakes require significant volumes of communication, RAM (message buffers) and computation.
- A solution will need to consider support for a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. Section 9.2).

8.3. Authentication

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

8.4. Server-side Authorization

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

8.5. Client-side Authorization Information

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

8.6. Server-side Authorization Information

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

8.7. Resource Access

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

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

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

8.8. Keys and Cipher Suites

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

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

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

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

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

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

8.9. Network Considerations

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

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

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

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

8.10. Legacy Considerations

- A solution may consider interworking with existing infrastructure.

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

9. Security Considerations

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

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

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

9.1. Physical Attacks on Sensor and Actuator Networks

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

The main threats to sensors and actuator networks are:

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

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

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

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

- Instead of manipulating the sensor data the attacker could change the physical object which the sensor is measuring, thereby changing the payload data which is being sent.

- Instead of manipulating data for an actuator or attacking the authorization system, the attacker could perform an unauthorized action directly on the physical object.

- A denial-of-service attack could be performed physically on the object or device.

  All these attacks are possible by having physical access to the device, since the assets are related to the physical world. Moreover, this kind of attacks are in many cases straightforward (requires no special competence or tools, low cost given physical access, etc.)

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

  Since it does not make sense to design a solution for a situation that cannot be protected against we assume there is no need to protect assets which are exposed during a physical attack. In other words, either an attacker does not have physical access to the sensor or actuator device, or if it has, the attack shall only have effect during the period of physical attack, and shall be limited in extent to the physical control the attacker exerts (e.g., must not affect the security of other devices.)

9.2. 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 a public key certificate, access token, or some other assertion, is valid.

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

For certain categories of devices we can assume that there is an internal clock which is sufficiently accurate to handle the time measurement requirements. If RS can connect directly to AS, 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, it may still be able to handle short lived access rights within some margins, by measuring the time since arrival of authorization information or request.

Some categories of devices in scope may be unable measure time with any accuracy (e.g. because of sleep cycles). This category of devices is not suitable for the use cases which require measuring validity of assertions and authorizations in terms of absolute time.

10. IANA Considerations

This document has no actions for IANA.

11. Informative References

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

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

[I-D.gerdes-ace-tasks]
Gerdes, S., "Authorization-Related Tasks in Constrained Environments", draft-gerdes-ace-tasks-00 (work in progress), September 2015.

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

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-05 (work in progress), July 2016.

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

[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.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <http://www.rfc-editor.org/info/rfc2904>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <http://www.rfc-editor.org/info/rfc4120>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[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>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[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>.

[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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7744>.

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
SICS Swedish ICT AB
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

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

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

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 15, 2017

M. Jones
Microsoft
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
ARM Ltd.

April 13, 2017

CBOR Web Token (CWT)
draft-ietf-ace-cbor-web-token-04

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on October 15, 2017.

Table of Contents

1. Introduction

The JSON Web Token (JWT) [RFC7519] is a standardized security token format that has found use in OAuth 2.0 and OpenID Connect deployments, among other applications. JWT uses JSON Web Signature (JWS) [RFC7515] and JSON Web Encryption (JWE) [RFC7516] to secure the contents of the JWT, which is a set of claims represented in JSON. The use of JSON for encoding information is popular for Web and native applications, but it is considered inefficient for some Internet of Things (IoT) systems that use low power radio technologies.

An alternative encoding of claims is defined in this document. Instead of using JSON, as provided by JWTs, this specification uses CBOR [RFC7049] and calls this new structure "CBOR Web Token (CWT)", which is a compact means of representing secured claims to be transferred between two parties. CWT is closely related to JWT. It references the JWT claims and both its name and pronunciation are derived from JWT. To protect the claims contained in CWTs, the CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] specification is used.

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document reuses terminology from JWT [RFC7519] and COSE [I-D.ietf-cose-msg].

StringOrURI:
: The "StringOrURI" term has the same meaning, syntax, and processing rules as the "StringOrUri" term defined in Section 2 of

JWT [RFC7519], except that a CWT StringOrURI uses CBOR major type 3 (text string) instead of a JSON string value.

NumericDate:
The "NumericDate" term has the same meaning, syntax, and processing rules as the "NumericDate" term defined in Section 2 of JWT [RFC7519], except that a CWT NumericDate uses one of the CBOR numeric types (0, 1, or 7 with subtypes 25, 26, or 27), instead of a numeric JSON value. The numeric date values that can used for a CWT NumericDate are identical to the epoch-based date/time values that are specified to follow the tag defined in Section 2.4.1 (Date and Time) of [RFC7049], except that the tag itself need not be present.

CBOR encoded claim key:
The key used to identify a claim value.

CWT Claims Set
A CBOR map that contains the claims conveyed by the CWT.

3. Claims

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

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

3.1. Claim Names

None of the claims defined below are intended to be mandatory to use or implement. They rather provide a starting point for a set of useful, interoperable claims. Applications using CWTs should define which specific claims they use and when they are required or optional.

3.1.1. iss (Issuer) Claim

The "iss" (issuer) claim has the same meaning, syntax, and processing rules as the "iss" claim defined in Section 4.1.1 of JWT [RFC7519], except that the format MUST be a StringOrURI. The CBOR encoded claim key 1 MUST be used to identify this claim.

3.1.2. sub (Subject) Claim

The "sub" (subject) claim has the same meaning, syntax, and processing rules as the "sub" claim defined in Section 4.1.2 of JWT [RFC7519], except that the format MUST be a StringOrURI. The CBOR encoded claim key 2 MUST be used to identify this claim.

3.1.3. aud (Audience) Claim

The "aud" (audience) claim has the same meaning, syntax, and processing rules as the "aud" claim defined in Section 4.1.3 of JWT [RFC7519], except that the format MUST be a StringOrURI. The CBOR encoded claim key 3 MUST be used to identify this claim.

3.1.4. exp (Expiration Time) Claim

The "exp" (expiration time) claim has the same meaning, syntax, and processing rules as the "exp" claim defined in Section 4.1.4 of JWT [RFC7519], except that the format MUST be a CWT NumericDate. The CBOR encoded claim key 4 MUST be used to identify this claim.

3.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim has the same meaning, syntax, and processing rules as the "nbf" claim defined in Section 4.1.5 of JWT [RFC7519], except that the format MUST be a CWT NumericDate. The CBOR encoded claim key 5 MUST be used to identify this claim.

3.1.6. iat (Issued At) Claim

The "iat" (issued at) claim has the same meaning, syntax, and processing rules as the "iat" claim defined in Section 4.1.6 of JWT [RFC7519], except that the format MUST be a CWT NumericDate. The CBOR encoded claim key 6 MUST be used to identify this claim.

3.1.7. cti (CWT ID) Claim

The "cti" (CWT ID) claim has the same meaning, syntax, and processing rules as the "jti" claim defined in Section 4.1.7 of JWT [RFC7519], except that the format MUST be of major type 2, binary string. The CBOR encoded claim key 7 MUST be used to identify this claim.

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

Claim	CBOR encoded claim key	CBOR major type of value
iss	1	3
sub	2	3
aud	3	3
exp	4	0, 1, or 7 with float subtype
nbf	5	0, 1, or 7 with float subtype
iat	6	0, 1, or 7 with float subtype
cti	7	2

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

5. CBOR Tags and Claim Values

The use of CBOR tags to prefix any of the claim values defined in this specification is NOT RECOMMENDED. For instance, while CBOR tag 6.1 (seconds-since-the-epoch) could logically be prefixed to values of the "exp", "nbf", and "iat" claims, this is unnecessary, since the representation of the claim values is already specified by the claim definitions. Tagging claim values would only take up extra space, without adding information. However, other claims defined by other specifications can specify that a tag prefix the claim value, when appropriate.

6. CWT CBOR Tag

How to determine that a CBOR data structure is a CWT is application-dependent. In some cases, this information is known from the application context, such as from the position of the CWT in a data structure at which the value must be a CWT. One method of indicating that a CBOR object is a CWT is the use of the "application/cwt" content type by a transport protocol.

This section defines the CWT CBOR tag as another means for applications to declare that a CBOR data structure is a CWT. Its use is optional, and is intended for use in cases in which this information would not otherwise be known.

If present, the CWT tag MUST prefix a tagged object using one of the COSE CBOR tags. In this example, the COSE_Mac0 tag is used. The actual COSE_Mac0 object has been excluded from this example.

```
/ CWT CBOR tag / 61(
  / COSE_Mac0 CBOR tag / 17(
    / COSE_Mac0 object /
  )
)
```

Figure 2: Example of a CWT tag usage

7. Creating and Validating CWTs

7.1. Creating a CWT

To create a CWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Create a CWT Claims Set containing the desired claims.

2. Let the Message be the binary representation of the CWT Claims Set.

3. Create a COSE Header containing the desired set of Header Parameters. The COSE Header MUST be valid per the [I-D.ietf-cose-msg] specification.

4. Depending upon whether the CWT is signed, MACed, or encrypted, there are three cases:

   * If the CWT is signed, create a COSE_Sign/COSE_Sign1 object using the Message as the COSE_Sign/COSE_Sign1 Payload; all steps specified in [I-D.ietf-cose-msg] for creating a COSE_Sign/COSE_Sign1 object MUST be followed.

   * Else, if the CWT is MACed, create a COSE_Mac/COSE_Mac0 object using the Message as the COSE_Mac/COSE_Mac0 Payload; all steps specified in [I-D.ietf-cose-msg] for creating a COSE_Mac/ COSE_Mac0 object MUST be followed.

   * Else, if the CWT is a COSE_Encrypt/COSE_Encrypt0 object, create a COSE_Encrypt/COSE_Encrypt0 using the Message as the plaintext for the COSE_Encrypt/COSE_Encrypt0 object; all steps specified in [I-D.ietf-cose-msg] for creating a COSE_Encrypt/ COSE_Encrypt0 object MUST be followed.

5. If a nested signing, MACing or encryption operation will be performed, let the Message be the COSE_Sign/COSE_Sign1, COSE_Mac/ COSE_Mac0 or COSE_Encrypt/COSE_Encrypt0, and return to Step 3, using a "content type" header value corresponding to the media

type "application/cwt" in the new COSE Header created in that step.

6. If needed by the application, add the appropriate COSE CBOR tag to the COSE object to indicate type of COSE object. If also needed by the application, add the CWT CBOR tag to indicate that the COSE object is a CWT.

7.2. Validating a CWT

When validating a CWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If any of the listed steps fail, then the CWT MUST be rejected -- that is, treated by the application as invalid input.

1. Verify that the CWT is a valid CBOR object.

2. If the object begins with the CWT CBOR tag, remove it and verify that one of the COSE CBOR tags follows it.

3. If the object is tagged with one of the COSE CBOR tags, remove it and verify that it corresponds to the structure of the following COSE object.

4. Verify that the resulting COSE Header includes only parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.

5. Use the CBOR tag to determine the type of the CWT, COSE_Sign/ COSE_Sign1, COSE_Mac/COSE_Mac0, or COSE_Encrypt/COSE_Encrypt0.

6. Depending upon whether the CWT is a signed, MACed, or encrypted, there are three cases:

   * If the CWT is a COSE_Sign/COSE_Sign1, follow the steps specified in [I-D.ietf-cose-msg] Section 4 (Signing Objects) for validating a COSE_Sign/COSE_Sign1 object. Let the Message be the COSE_Sign/COSE_Sign1 payload.

   * Else, if the CWT is a COSE_Mac/COSE_Mac0, follow the steps specified in [I-D.ietf-cose-msg] Section 6 (MAC Objects) for validating a COSE_Mac/COSE_Mac0 object. Let the Message be the COSE_Mac/COSE_Mac0 payload.

   * Else, if the CWT is a COSE_Encrypt/COSE_Encrypt0 object, follow the steps specified in [I-D.ietf-cose-msg] Section 5

(Encryption Objects) for validating a COSE_Encrypt/ COSE_Encrypt0 object. Let the Message be the resulting plaintext.

7. If the COSE Header contains a "content type" header value corresponding to the media type "application/cwt", then the Message is a CWT that was the subject of nested signing or encryption operations. In this case, return to Step 1, using the Message as the CWT.

8. Verify that the Message is a valid CBOR object; let the CWT Claims Set be this CBOR object.

8. Security Considerations

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

9. IANA Considerations

9.1. CBOR Web Token (CWT) Claims Registry

This section establishes the IANA "CBOR Web Token (CWT) Claims" registry.

Values are registered on a Specification Required [RFC5226] basis, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published.

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration description is clear.

9.1.1. Registration Template

Claim Name:
: The human-readable name requested (e.g., "iss").

Claim Description:
: Brief description of the claim (e.g., "Issuer").

JWT Claim Name:
: Claim Name of the equivalent JWT claim, as registered in [IANA.JWT.Claims]. CWT claims should normally have a corresponding JWT claim. If a corresponding JWT claim would not make sense, the Designated Experts can choose to accept registrations for which the JWT Claim Name is listed as "N/A".

CBOR Key Value:
: Key value for the claim. The key value MUST be an integer in the range of 1 to 65536.

CBOR Major Type:
: CBOR major type and optional tag for the claim.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

9.1.2. Initial Registry Contents

- Claim Name: "iss"
- Claim Description: Issuer
- JWT Claim Name: "iss"
- CBOR Key Value: 1
- CBOR Major Type: 3
- Change Controller: IESG
- Specification Document(s): Section 3.1.1 of [[this specification]]

- Claim Name: "sub"
- Claim Description: Subject
- JWT Claim Name: "sub"
- CBOR Key Value: 2

- CBOR Major Type: 3
- Change Controller: IESG
- Specification Document(s): Section 3.1.2 of [[this specification]]

- Claim Name: "aud"
- Claim Description: Audience
- JWT Claim Name: "aud"
- CBOR Key Value: 3
- CBOR Major Type: 3
- Change Controller: IESG
- Specification Document(s): Section 3.1.3 of [[this specification]]

- Claim Name: "exp"
- Claim Description: Expiration Time
- JWT Claim Name: "exp"
- CBOR Key Value: 4
- CBOR Major Type: 0, 1, or 7 with subtypes 25, 26, or 27
- Change Controller: IESG
- Specification Document(s): Section 3.1.4 of [[this specification]]

- Claim Name: "nbf"
- Claim Description: Not Before
- JWT Claim Name: "nbf"
- CBOR Key Value: 5
- CBOR Major Type: 0, 1, or 7 with subtypes 25, 26, or 27
- Change Controller: IESG
- Specification Document(s): Section 3.1.5 of [[this specification]]

- Claim Name: "iat"
- Claim Description: Issued At
- JWT Claim Name: "iat"
- CBOR Key Value: 6
- CBOR Major Type: 0, 1, or 7 with subtypes 25, 26, or 27
- Change Controller: IESG
- Specification Document(s): Section 3.1.6 of [[this specification]]

- Claim Name: "cti"
- Claim Description: CWT ID
- JWT Claim Name: "jti"
- CBOR Key Value: 7
- CBOR Major Type: 2
- Change Controller: IESG

- Specification Document(s): Section 3.1.7 of [[this specification]]

9.2. Media Type Registration

This section registers the "application/cwt" media type in the "Media Types" registry [IANA.MediaTypes] in the manner described in RFC 6838 [RFC6838], which can be used to indicate that the content is a CWT.

9.2.1. Registry Contents

- Type name: application
- Subtype name: cwt
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See the Security Considerations section of [[this specification]]
- Interoperability considerations: N/A
- Published specification: [[this specification]]
- Applications that use this media type: IoT applications sending security tokens over HTTP(S) and other transports.
- Fragment identifier considerations: N/A
- Additional information:

  Magic number(s): N/A
  File extension(s): N/A
  Macintosh file type code(s): N/A

- Person & email address to contact for further information: IESG, iesg@ietf.org
- Intended usage: COMMON
- Restrictions on usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change controller: IESG
- Provisional registration? No

9.3. CoAP Content-Formats Registration

This section registers the CoAP Content-Format ID for the "application/cwt" media type in the "CoAP Content-Formats" registry [IANA.CoAP.Content-Formats].

9.3.1. Registry Contents

- Media Type: application/cwt
- Encoding: -
- Id: TBD (maybe 61)

- Reference: [[this specification]]

9.4. CBOR Tag registration

This section registers the CWT CBOR tag in the "CBOR Tags" registry [IANA.CBOR.Tags].

9.4.1. Registry Contents

- CBOR Tag: TBD (maybe 61 to use the same value as the Content-Format)
- Data Item: CBOR Web Token (CWT)
- Semantics: CBOR Web Token (CWT), as defined in [[this specification]]
- Reference: [[this specification]]
- Point of Contact: Michael B. Jones, mbj@microsoft.com

10. References

10.1. Normative References

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

[IANA.CBOR.Tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>.

[IANA.CoAP.Content-Formats]
IANA, "CoAP Content-Formats", <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

[IANA.MediaTypes]
IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[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>.

[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>.

10.2. Informative References

[IANA.JWT.Claims] IANA, "JSON Web Token Claims", <http://www.iana.org/assignments/jwt>.

[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>.

[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>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

Appendix A. Examples

This appendix includes a set of CWT examples that show how the CWT Claims Set can be protected. There are examples that are signed, MACed, encrypted, and that use nested signing and encryption. To make the examples easier to read, they are presented both as hex strings and in the extended CBOR diagnostic notation described in Section 6 of [RFC7049].

A.1. Example CWT Claims Set

The CWT Claims Set used for the different examples displays usage of all the defined claims. For signed and MACed examples, the CWT Claims Set is the CBOR encoding as a binary string.

```
a70175636f61703a2f2f61732e6578616d706c652e636f6d02656572696b7703
7818636f61703a2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0
051a5610d9f0061a5610d9f007420b71
```

Figure 3: Example CWT Claims Set as hex string

```
{
  / iss / 1: "coap://as.example.com",
  / sub / 2: "erikw",
  / aud / 3: "coap://light.example.com",
  / exp / 4: 1444064944,
  / nbf / 5: 1443944944,
  / iat / 6: 1443944944,
  / cti / 7: h'0b71'
}
```

Figure 4: Example CWT Claims Set in CBOR diagnostic notation

A.2. Example keys

This section contains the keys used to sign, MAC, and encrypt the messages in this appendix. Line breaks are for display purposes only.

A.2.1. 128-bit Symmetric Key as Hex Encoded String

```
8e82e68e61654ecb5a369fe8be7572dd
```

A.2.2. 256-bit Symmetric Key as Hex Encoded String

```
403697de87af64611c1d32a05dab0fe1fcb715a86ab435f1ec99192d79569388
```

A.2.3. ECDSA P-256 256-bit COSE Key

```
a622582060f7f1a780d8a783bfb7a2dd6b2796e8128dbbcef9d3d168db952997
1a36e7b92358206c1382765aec5358f117733d281c1c7bdc39884d04a45a1e6c
67c858bc206c19032601022158201433 29cce7868e416927599cf65a34f3ce2f
fda55a7eca69ed8919a394d42f0f2001
```

Figure 5: ECDSA 256-bit COSE Key as hex string

```
{
  / d /   -4: h'6c1382765aec5358f117733d281c1c7bdc39884d04a45a1e
              6c67c858bc206c19',
  / y /   -3: h'60f7f1a780d8a783bfb7a2dd6b2796e8128dbbcef9d3d168
              db9529971a36e7b9',
  / x /   -2: h'143329cce7868e416927599cf65a34f3ce2ffda55a7eca69
              ed8919a394d42f0f',
  / crv / -1: 1 / P-256 / ,
  / kty /  1: 2 / EC2 / ,
  / alg /  3: -7 / ECDSA 256 /
}
```

Figure 6: ECDSA 256-bit COSE Key in CBOR diagnostic notation

A.3. Example Signed CWT

This section shows a signed CWT with a single recipient and a full CWT Claims Set.

The signature is generated using the private key listed in Appendix A.2.3 and it can be validated using the public key from Appendix A.2.3. Line breaks are for display purposes only.

```
d28443a10126a05850a70175636f61703a2f2f61732e6578616d706c652e636f6
d02656572696b77037818636f61703a2f2f6c696768742e6578616d706c652e63
6f6d041a5612aeb0051a5610d9f0061a5610d9f007420b7158401fe410abce650
effed497f05d7f9462de67d571384097de0d96f1e2514d284cdd85634f269af6c
36c64f22e7691abb464bed2ff23176cdba9fd9e213f637d082
```

Figure 7: Signed CWT as hex string

```
18(
  [
    / protected / h'a10126' / {
      / alg / 1: -7 / ECDSA 256 /
    } / ,
    / unprotected / {},
    / payload / h'a70175636f61703a2f2f61732e6578616d706c652e63
                  6f6d02656572696b77037818636f61703a2f2f6c6967
                  68742e6578616d706c652e636f6d041a5612aeb0051a
                  5610d9f0061a5610d9f007420b71' / {
        / iss / 1: "coap://as.example.com",
        / sub / 2: "erikw",
        / aud / 3: "coap://light.example.com",
        / exp / 4: 1444064944,
        / nbf / 5: 1443944944,
        / iat / 6: 1443944944,
        / cti / 7: h'0b71'
      } / ,
    / signature / h'1fe410abce650effed497f05d7f9462de67d571384
                    097de0d96f1e2514d284cdd85634f269af6c36c64f
                    22e7691abb464bed2ff23176cdba9fd9e213f637d0
                    82'
  ]
)
```

Figure 8: Signed CWT in CBOR diagnostic notation

A.4. Example MACed CWT

This section shows a MACed CWT with a single recipient and a full CWT Claims Set.

The MAC is generated using the 256-bit symmetric key from Appendix A.2.2 with a 64-bit truncation. Line breaks are for display purposes only.

```
d83dd18443a10104a05850a70175636f61703a2f2f61732e6578616d706c652e
636f6d02656572696b77037818636f61703a2f2f6c696768742e6578616d706c
652e636f6d041a5612aeb0051a5610d9f0061a5610d9f007420b7148093101ef
6d789200
```

Figure 9: MACed CWT with CWT tag as hex string

```
61(
  17(
    [
      / protected / h'a10104' / {
        / alg / 1: 4 / HMAC 256/64 /
      } / ,
      / unprotected / {},
      / payload / h'a70175636f61703a2f2f61732e6578616d706c652e636f
                    6d02656572696b77037818636f61703a2f2f6c69676874
                    2e6578616d706c652e636f6d041a5612aeb0051a5610d9
                    f0061a5610d9f007420b71' / {
          / iss / 1: "coap://as.example.com",
          / sub / 2: "erikw",
          / aud / 3: "coap://light.example.com",
          / exp / 4: 1444064944,
          / nbf / 5: 1443944944,
          / iat / 6: 1443944944,
          / cti / 7: h'0b71'
        } / ,
      / tag / h'093101ef6d789200'
    ]
  )
)
```

Figure 10: MACed CWT with CWT tag in CBOR diagnostic notation

A.5. Example Encrypted CWT

This section shows an encrypted CWT with a single recipient and a full CWT Claims Set.

The encryption is done with AES-CCM mode using the 128-bit symmetric key from Appendix A.2.1 with a 64-bit tag and 13-byte nonce, i.e., COSE AES-CCM-16-64-128. Line breaks are for display purposes only.

```
d08343a1010aa1054d3a869e378e72b77d077c29be025858d275ad9cd7df1b10
ba8cde785c74b1e1e6ada287e2baf1451b06862529b784d230b0111773b6c369
1319aec4dcc379fe47115a5d62632727c05f4567fc84dd79554db86676a14978
42de805d8be93180af4d6ff3043886a0
```

Figure 11: Encrypted CWT as hex string

```
16(
  [
    / protected / h'a1010a' / {
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } /,
    / unprotected / {
      / iv / 5: h'3a869e378e72b77d077c29be02'
    },
    / ciphertext / h'd275ad9cd7df1b10ba8cde785c74b1e1e6ada287e2b
                    af1451b06862529b784d230b0111773b6c3691319ae
                    c4dcc379fe47115a5d62632727c05f4567fc84dd795
                    54db86676a1497842de805d8be93180af4d6ff30438
                    86a0'
  ]
)
```

Figure 12: Encrypted CWT in CBOR diagnostic notation

A.6. Example Nested CWT

This section shows a Nested CWT, signed and then encrypted, with a single recipient and a full CWT Claims Set.

The signature is generated using the private ECDSA key from Appendix A.2.3 and it can be validated using the public ECDSA parts from Appendix A.2.3. The encryption is done with AES-CCM mode using the 128-bit symmetric key from Appendix A.2.1 with a 64-bit tag and 13-byte nonce, i.e., COSE AES-CCM-16-64-128. The content type is set to CWT to indicate that there are multiple layers of COSE protection before finding the CWT Claims Set. The decrypted ciphertext will be a COSE_sign1 structure. In this example, it is the same one as in Appendix A.3, i.e., a Signed CWT Claims Set. Note that there is no limitation to the number of layers; this is an example with two layers. Line breaks are for display purposes only.

```
d08346a203183d010aa1054d9120e5dc42c9f9aec05ebe8a4858a538be026c02
4a40b19d6dbea3ddb18b31021f874a097a05ff3cdaa4665bafc8e46a3d7f37ad
f002fe57eee267f8f62a9c1621af75e1ecd742a3d801c2cc82358cf104a8d902
4d38a599ea6027d482dc2948a88fe83f9734804299c832401029e2d32a984789
c8e9563e8d2a751323bb7e4462b549e0fa89ef93f78bf6425635fba76b4aa804
7908e89b3b7c3d59d8a80e22f70a1b6ee8c162c564341c2f15cec252d3da038c
```

Figure 13: Signed and Encrypted CWT as hex string

```
16(
  [
    / protected / h'a203183d010a' / {
      / content type / 3: 61, / CWT /
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } / ,
    / unprotected / {
      / iv / 5: h'9120e5dc42c9f9aec05ebe8a48'
    },
    / ciphertext / h'38be026c024a40b19d6dbea3ddb18b31021f874a097
                     a05ff3cdaa4665bafc8e46a3d7f37adf002fe57eee2
                     67f8f62a9c1621af75e1ecd742a3d801c2cc82358cf
                     104a8d9024d38a599ea6027d482dc2948a88fe83f97
                     34804299c832401029e2d32a984789c8e9563e8d2a7
                     51323bb7e4462b549e0fa89ef93f78bf6425635fba7
                     6b4aa8047908e89b3b7c3d59d8a80e22f70a1b6ee8c
                     162c564341c2f15cec252d3da038c'
  ]
)
```

Figure 14: Signed and Encrypted CWT in CBOR diagnostic notation

Appendix B. Acknowledgements

This specification is based on JSON Web Token (JWT) [RFC7519], the authors of which also include Nat Sakimura and John Bradley. Ludwig Seitz and Goeran Selander have made contributions the specification.

Appendix C. Document History

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

-04

- Specified that the use of CBOR tags to prefix any of the claim values defined in this specification is NOT RECOMMENDED.

-03

- Reworked the examples to include signed, MACed, encrypted, and nested CWTs.
- Defined the CWT CBOR tag and explained its usage.

-02

- Added IANA registration for the application/cwt media type.
- Clarified the nested CWT language.
- Corrected nits identified by Ludwig Seitz.

-01

- Added IANA registration for CWT Claims.
- Added IANA registration for the application/cwt CoAP content-format type.
- Added Samuel Erdtman as an editor.
- Changed Erik's e-mail address.

-00

- Created the initial working group version based on draft-wahlstroem-ace-cbor-web-token-00.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

Samuel Erdtman
Spotify AB
Birger Jarlsgatan 61, 4tr
Stockholm 113 56
Sweden

Phone: +46702691499
Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

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

L. Seitz
RISE SICS
G. Selander
Ericsson
E. Wahlstroem
(no affiliation)
S. Erdtman
Spotify AB
H. Tschofenig
ARM Ltd.

March 13, 2017

Authentication and Authorization for Constrained Environments (ACE) draft-ietf-ace-oauth-authz-06

Abstract

This specification defines a framework for authentication and authorization in Internet of Things (IoT) environments. The framework is based on a set of building blocks including OAuth 2.0 and CoAP, thus making a well-known and widely used authorization solution suitable for IoT devices. Existing specifications are used where possible, but where the constraints of IoT devices require it, extensions are added and profiles are defined.

Table of Contents

1. Introduction

Authorization is the process for granting approval to an entity to access a resource [RFC4949]. The authorization task itself can best be described as granting access to a requesting client, for a resource hosted on a device, the resource server (RS). This exchange is mediated by one or multiple authorization servers (AS). Managing authorization for a large number of devices and users is a complex task.

While prior work on authorization solutions for the Web and for the mobile environment also applies to the IoT environment many IoT devices are constrained, for example in terms of processing capabilities, available memory, etc. For web applications on constrained nodes this specification makes use of CoAP [RFC7252].

A detailed treatment of constraints can be found 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-powered devices, and all levels in between.

Hence, IoT devices may be very different in terms of available processing and message exchange capabilities and there is a need to support many different authorization use cases [RFC7744].

This specification describes a framework for authentication and authorization in constrained environments (ACE) built on re-use of OAuth 2.0 [RFC6749], thereby extending authorization to Internet of Things devices. This specification contains the necessary building blocks for adjusting OAuth 2.0 to IoT environments.

More detailed, interoperable specifications can be found in profiles. Implementations may claim conformance with a specific profile, whereby implementations utilizing the same profile interoperate while implementations of different profiles are not expected to be interoperable. Some devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of low end devices.

2. Terminology

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

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as /token and /introspect at the AS and /authz-info at the RS. The CoAP [RFC7252] definition, which is "An entity participating in the CoAP protocol" is not used in this memo.

Since this specification 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]).

We call the specifications of this memo the "framework" or "ACE framework". When referring to "profiles of this framework" we mean additional memo's that define the use of this specification with concrete transport, and communication security protocols (e.g. CoAP over DTLS).

3. Overview

This specification defines the ACE framework for authorization in the Internet of Things environment. It consists 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. While this specification defines extensions for the use of OAuth over CoAP, we do envision further underlying protocols to be supported in the future, such as HTTP/2, MQTT and QUIC.

A third building block is CBOR [RFC7049] for encodings where JSON [RFC7159] is not sufficiently compact. CBOR is a binary encoding designed for small code and message size, which may be used for encoding of self contained tokens, and also for encoding CoAP POST parameters and CoAP responses.

A fourth building block is the compact CBOR-based secure message format COSE [I-D.ietf-cose-msg], which enables application layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC5246]). COSE is used to secure self contained tokens such as proof-of-possession (PoP) tokens, which is

an extension to the OAuth access tokens, and "client tokens" which are defined in this framework (see Section 5.6.4). The default access token format is defined in CBOR web token (CWT) [I-D.ietf-ace-cbor-web-token]. Application layer security for CoAP using COSE can be provided with OSCOAP [I-D.ietf-core-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 ACE framework nevertheless takes all these aspects into account and allows several different deployment variants to co-exist rather than mandating a one-size-fits-all solution. We believe this is important to cover the wide range of possible interworking use cases and the different requirements from a security point of view. Once IoT deployments mature, popular deployment variants will be documented in form of ACE 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 information, or references to it, 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 specification:

The token and introspect Endpoints:

The AS hosts the /token endpoint that allows a client to request access tokens. The client makes a POST request to the /token endpoint on the AS and receives the access token in the response (if the request was successful).

The token introspection endpoint, /introspect, is used by the RS when requesting additional information regarding a received access token. The RS makes a POST request to /introspect on the AS and

receives information about the access token in the response. (See "Introspection" below.)

Access Tokens:

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

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 bound to the access token. When this specification uses the term "access token" it is assumed to be a PoP token unless specifically stated otherwise.

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

Symmetric PoP key: The AS generates a random symmetric PoP key. The key is either stored to be returned on introspection calls or encrypted and included in the 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 either stored to be returned on introspection calls or included inside the access token and sent back to the requesting client. The RS can identify the client's public key from the information in the token, which allows the client to use the corresponding private key for the proof of possession.

The access token is either a simple reference, or a structured information object (e.g. CWT [I-D.ietf-ace-cbor-web-token]), protected by a cryptographic wrapper (e.g. COSE [I-D.ietf-cose-msg]). The choice of PoP key does not necessarily imply a specific credential type for the integrity protection of the token.

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 token request. In turn, the AS may use the scope response parameter to inform the client of the scope of the access token issued. As the client could be a constrained device as well, this specification uses CBOR encoded messages for CoAP, defined in Section 5, 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:

Information carried in the access token or returned from introspection, called claims, is in the form of type-value pairs. An access token may, for example, include a claim identifying the AS that issued the token (via the "iss" claim) and what audience the access token is intended for (via 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 what claims are put into the access token by the authorization server.

While the structure and encoding of the access token varies throughout deployments, a standardized format has been defined with the JSON Web Token (JWT) [RFC7519] where claims are encoded as a JSON object. In [I-D.ietf-ace-cbor-web-token] an equivalent format using CBOR encoding (CWT) has been defined.

Introspection:

Introspection is a method for a resource server to query the authorization server for the active state and content of a received access token. This is particularly useful in those cases where the authorization decisions are very dynamic and/or where the received access token itself is a reference rather than a self-contained token. More information about introspection in OAuth 2.0 can be found in [RFC7662].

3.2. CoAP

CoAP is an application layer protocol similar to HTTP, but specifically designed for constrained environments. CoAP typically uses datagram-oriented transport, such as UDP, where reordering and loss of packets can occur. A security solution need to take the latter aspects into account.

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

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [RFC7959]. However, block-wise transfer does not increase the size limits of CoAP options, therefore data encoded in options has to be kept small.

Transport layer security for CoAP can be provided by DTLS 1.2 [RFC6347] or TLS 1.2 [RFC5246]. CoAP defines a number of proxy operations which requires transport layer security to be terminated at the proxy. One approach for protecting CoAP communication end-to-end through proxies, and also to support security for CoAP over a different transport in a uniform way, is to provide security on application layer using an object-based security mechanism such as COSE [I-D.ietf-cose-msg].

One application of COSE is OSCOAP [I-D.ietf-core-object-security], which provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages. In OSCOAP, the CoAP messages are wrapped in COSE objects and sent using CoAP.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the /token and /introspect endpoints. A client obtains an access token from an AS using the /token endpoint and subsequently presents the access token to a RS to gain access to a protected resource. The RS, after receiving an access token, may present it to the AS via the /introspect endpoint to get information about the

access token. In other deployments the RS may process the access token locally without the need to contact an AS. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in Section 3.1.

The OAuth 2.0 framework defines a number of "protocol flows" via grant types, which have been extended further with extensions to OAuth 2.0 (such as RFC 7521 [RFC7521] and [I-D.ietf-oauth-device-flow]). What grant types works best depends on the usage scenario and RFC 7744 [RFC7744] describes many different IoT use cases but there are two preferred grant types, namely the Authorization Code Grant (described in Section 4.1 of RFC 7521) and the Client Credentials Grant (described in Section 4.4 of RFC 7521). The Authorization Code Grant is a good fit for use with apps running on smart phones and tablets that request access to IoT devices, a common scenario in the smart home environment, where users need to go through an authentication and authorization phase (at least during the initial setup phase). The native apps guidelines described in [I-D.ietf-oauth-native-apps] are applicable to this use case. The Client Credential Grant is a good fit for use with IoT devices where the OAuth client itself is constrained. In such a case the resource owner or another person on his or her behalf have arranged with the authorization server out-of-band, which is often accomplished using a commissioning tool.

The consent of the resource owner, for giving a client access to a protected resource, can be provided dynamically as in the traditional OAuth flows, or it could be pre-configured by the resource owner as authorization policies at the AS, which the AS evaluates when a token request arrives. The resource owner and the requesting party (i.e. client owner) are not shown in Figure 1.

This framework supports a wide variety of communication security mechanisms between the ACE entities, such as client, AS, and RS. We assume that the client has been registered (also called enrolled or onboarded) to an AS using a mechanism defined outside the scope of this document. In practice, various techniques for onboarding have been used, such as factory-based provisioning or the use of commissioning tools. Regardless of the onboarding technique, this registration procedure implies that the client and the AS share credentials, and configuration parameters. These credentials are used to mutually authenticate each other and to protect messages exchanged between the client and the AS.

It is also assumed that the RS has been registered with the AS, potentially in a similar way as the client 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 its 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 detailed procedures for this discovery process may be defined in an ACE profile and depend on the protocols being used and the specific deployment environment.

In Bluetooth Low Energy, for example, advertisements are broadcasted by a peripheral, including information about the primary 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].

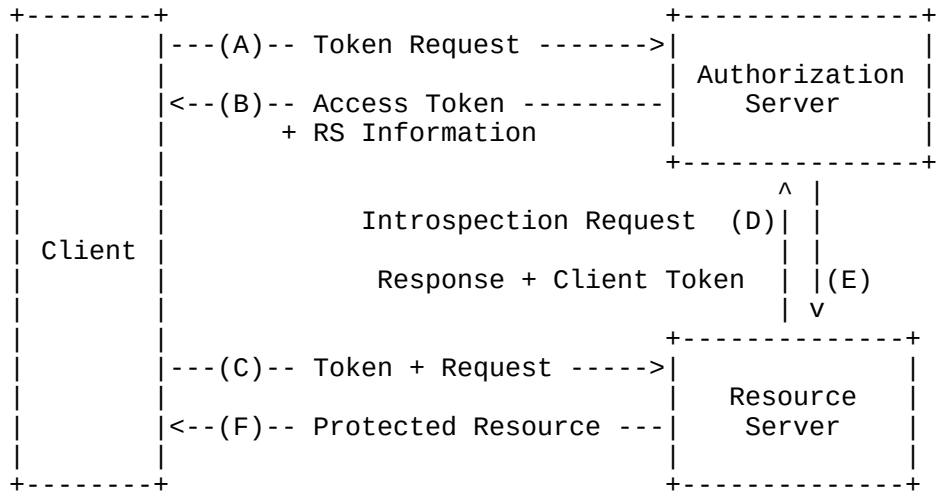


Figure 1: Basic Protocol Flow.

Requesting an Access Token (A):

The client makes an access token request to the /token endpoint at the AS. This framework 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 credentials it wants to use

(e.g., symmetric/asymmetric cryptography or a reference to a specific credential).

Access Token Response (B):

If the AS successfully processes the request from the client, it returns an access token. It also returns various parameters, referred as "RS Information". In addition to the response parameters defined by OAuth 2.0 and the PoP token extension, further response parameters, such as information on which profile the client should use with the resource server(s). More information about these parameters can be found in Section 5.5.4.

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, QUIC, MQTT, Bluetooth Low Energy, etc., are also viable candidates.

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

(1) the client sends the access token containing, or referencing, the authorization information to the RS, that may be used for subsequent resource requests by the client, and (2) the client makes the resource access request, using the communication security protocol and other RS Information obtained from the AS.

The Client and the RS mutually authenticate using the security protocol specified in the profile (see step B) and the keys obtained in the access token or the RS Information or the client token. 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 introspect the access token by including it in a request to the /introspect endpoint at that AS. Token introspection over CoAP is defined in Section 5.6 and for HTTP in [RFC7662].

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 most recent parameters, such as scope, audience, validity etc. associated with it back to the RS. The RS then uses the received parameters to process the request to either accept or to deny it. The AS can additionally return information that the RS needs to pass on to the client in the form of a client token. The latter is used to establish keys for mutual authentication between client and RS, when the client has no direct connectivity to the AS, see Section 5.6.4 for details.

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. Framework

The following sections detail the profiling and extensions of OAuth 2.0 for constrained environments which constitutes the ACE framework.

Credential Provisioning

For IoT we cannot generally assume that the client and RS are part of a common key infrastructure, so the AS provisions credentials or associated information to allow mutual authentication. These credentials need to be provided to the parties before or during the authentication protocol is executed, and may be re-used for subsequent token requests.

Proof-of-Possession

The ACE framework by default implements proof-of-possession for access tokens, i.e. that the token holder can prove being a holder of the key bound to the token. The binding is provided by the "cnf" claim indicating what key is used for proof-of-possession. If clients need to update a token, e.g. to get additional rights, they can request that the AS binds the new access token to the same key as the previous token.

ACE Profiles

The client or RS may be limited in the encodings or protocols it supports. To support a variety of different deployment settings, specific interactions between client and RS are defined in an ACE profile. In ACE framework the AS is expected to manage the matching of compatible profile choices between a client and an RS. The AS informs the client of the selected profile using the "profile" parameter in the token response.

OAuth 2.0 requires the use of TLS both to protect the communication between AS and client when requesting an access token; between client and RS when accessing a resource and between AS and RS for introspection. In constrained settings TLS is not always feasible, or desirable. Nevertheless it is REQUIRED that the data exchanged with the AS is encrypted and integrity protected. It is furthermore REQUIRED that the AS and the endpoint communicating with it (client or RS) perform mutual authentication.

Profiles MUST specify how mutual authentication is done, depending e.g. on the communication protocol and the credentials used by the client or the RS.

In OAuth 2.0 the communication with the Token and the Introspection endpoints at the AS is assumed to be via HTTP and may use Uri-query parameters. This framework RECOMMENDS to use CoAP instead and RECOMMENDS the use of the following alternative instead of Uri-query parameters: The sender (client or RS) encodes the parameters of its request as a CBOR map and submits that map as the payload of the POST request. The Content-format depends on the security applied to the content and MUST be specified by the profile that is used.

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. This framework RECOMMENDS the use of CBOR [RFC7049] instead. The requesting device can explicitly request this encoding by setting the CoAP Accept option in the request to "application/cbor". Depending on the profile, the content MAY arrive in a different format wrapping a CBOR payload.

5.1. Authorization Grants

To request an access token, the client obtains authorization from the resource owner or uses its client credentials as grant. The authorization is expressed in the form of an authorization grant.

The OAuth framework defines four grant types. The grant types can be split up into two groups, those granted on behalf of the resource owner (password, authorization code, implicit) and those for the client (client credentials).

The grant type selected depending based on the use case. In cases where the client will act on behalf of the resource owner, authorization code grant is recommended. If the client should to act on be half of the user but does not have any display or very limited interaction possibilities it is recommended to use the device code grant defined in [I-D.ietf-oauth-device-flow]. In cases where the client will not act on be half of the resource owner, client credentials grant is recommended.

For details on the different grant types see the OAuth 2.0 framework. The OAuth 2.0 framework provides an extension mechanism for defining additional grant types so profiles of this framework MAY define additional grant types if needed.

5.2. Client Credentials

Authentication of the client is mandatory independent of the grant type when requesting the access token from the token endpoint. In the case of client credentials grant type the authentication and grant coincides.

Client registration and provisioning of client credentials to the client is out of scope for this specification.

The OAuth framework, [RFC6749], defines one client credential type, client id and client secret. Profiles of this framework MAY extend with additional client credentials such as DTLS pre-shared keys or client certificates.

5.3. AS Authentication

Client credential does not by default authenticate the AS that the client connects to. In classic OAuth the AS is authenticated with a TLS server certificate.

Profiles of this framework SHOULD specify how clients authenticate the AS and how communication security is implemented, otherwise server side TLS certificates as defined by OAuth 2.0 is required.

5.4. The ’Authorize’ Endpoint

The authorization endpoint is used to interact with the resource owner and obtain an authorization grant. It is used for authorization code and implicit grant, flows that requires online user consent. In the most common implementation a users-agent is used and redirected from the client to the AS. The AS shows a consent dialog and directs back to the client, with the approved grant or an error message.

The grant types defined in OAuth 2.0, that use the authorization endpoint, require the use of a user agent (i.e. a browser). This endpoint is therefore out of scope for this specification. Implementations should use the definition and recommendations of [RFC6749] and [RFC6819].

If clients involved cannot support HTTP and TLS profiles MAY define mappings for the authorization endpoint.

5.5. The 'Token' Endpoint

In plain OAuth 2.0 the AS provides the /token endpoint for submitting access token requests. This framework extends the functionality of the /token endpoint, giving the AS the possibility to help client and RS to establish shared keys or to exchange their public keys. Furthermore this framework defines encodings using CoAP and CBOR, in addition to HTTP and JSON.

For the AS to be able to issue a token the client MUST be authenticated and present a valid grant for the scopes requested.

The figures of this section uses CBOR diagnostic notation without the integer abbreviations for the parameters or their values for better readability.

5.5.1. Client-to-AS Request

The client sends a CoAP POST request to the token endpoint at the AS, the profile MUST specify the Content-Type and wrapping of the payload. The content of the request consists of the parameters specified in section 4 of the OAuth 2.0 specification [RFC6749] encoded as a CBOR map.

In addition to these parameters, this framework defines the following parameters for requesting an access token from a /token endpoint:

aud
: OPTIONAL. Specifies the audience for which the client is requesting an access token. If this parameter is missing it is assumed that the client and the AS have a pre-established understanding of the audience that an access token should address. If a client submits a request for an access token without specifying an "aud" parameter, and the AS does not have a default "aud" value for this client, then the AS MUST respond with an error message with the CoAP response code 4.00 (Bad Request).

cnf

OPTIONAL. This field contains information about the key the client would like to bind to the access token for proof-of-possession. It is NOT RECOMMENDED that a client submits a symmetric key value to the AS using this parameter. See Section 5.5.4.5 for more details on the formatting of the 'cnf' parameter.

The following examples illustrate different types of requests for proof-of-possession tokens.

Figure 2 shows a request for a token with a symmetric proof-of-possession key. Note that in this example we assume a DTLS-based communication security profile, therefore the Content-Type is "application/cbor". The content is displayed in CBOR diagnostic notation, without abbreviations for better readability.

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "aud" : "tempSensor4711",
 }
```

Figure 2: Example request for an access token bound to a symmetric key.

Figure 3 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example we assume an object security-based profile, therefore the Content-Type is "application/cose+cbor".

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cose+cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id" : "myclient",
  "client_secret" : "mysecret234",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h'11',
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
    }
  }
}
```

Figure 3: Example request for an access token bound to an asymmetric key.

Figure 4 shows a request for a token where a previously communicated proof-of-possession key is only referenced. Note that we assume a DTLS-based communication security profile for this example, therefore the Content-Type is "application/cbor". Also note that the client performs a password based authentication in this example by submitting its client_secret (see section 2.3.1. of [RFC6749]).

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id" : "myclient",
  "client_secret" : "mysecret234",
  "aud" : "valve424",
  "scope" : "read",
  "cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

Figure 4: Example request for an access token bound to a key reference.

5.5.2. AS-to-Client Response

If the access token request has been successfully verified by the AS and the client is authorized to obtain an access token corresponding to its access token request, the AS sends a response with the CoAP response code 2.01 (Created). If client request was invalid, or not authorized, the AS returns an error response as described in Section 5.5.3.

Note that the AS decides which token type and profile to use when issuing a successful response. It is assumed that the AS has prior knowledge of the capabilities of the client, and the RS (see Appendix D. This prior knowledge may, for example, be set by the use of a dynamic client registration protocol exchange [RFC7591].

The content of the successful reply is the RS Information. It MUST be encoded as CBOR map, containing parameters as specified in section 5.1 of [RFC6749]. In addition to these parameters, the following parameters are also part of a successful response:

profile
: REQUIRED. This indicates the profile that the client MUST use towards the RS. See Section 5.5.4.4 for the formatting of this parameter.

cnf
: REQUIRED if the token type is 'pop'. OPTIONAL otherwise. If a symmetric proof-of-possession algorithms was selected, this field contains the proof-of-possession key. If an asymmetric algorithm was selected, this field contains information about the public key used by the RS to authenticate. See Section 5.5.4.5 for the formatting of this parameter.

token_type
: OPTIONAL. By default implementations of this framework SHOULD assume that the token_type is 'pop'. If a specific use case requires another token_type (e.g. 'Bearer') to be used then this parameter is REQUIRED.

Note that if CBOR Web Tokens [I-D.ietf-ace-cbor-web-token] are used, the access token can also contain a 'cnf' claim. This claim is however consumed by a different party. The access token is created by the AS and processed by the RS (and opaque to the client) whereas the RS Information is created by the AS and processed by the client; it is never forwarded to the resource server.

Figure Figure 5 summarizes the parameters that may be part of the RS Information.

Parameter name	Specified in
access_token	RFC 6749
token_type	RFC 6749
expires_in	RFC 6749
refresh_token	RFC 6749
scope	RFC 6749
state	RFC 6749
profile	[[this specification]]
cnf	[[this specification]]

Figure 5: RS Information parameters

The following examples illustrate different types of responses for proof-of-possession tokens.

Figure 6 shows a response containing a token and a 'cnf' parameter with a symmetric proof-of-possession key. Note that we assume a DTLS-based communication security profile for this example, therefore the Content-Type is "application/cbor".

```
Header: Created (Code=2.01)
Content-Type: "application/cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (remainder of CWT omitted for brevity;
   CWT contains COSE_Key in the 'cnf' claim)',
  "profile" : "coap_dtls",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 6: Example AS response with an access token bound to a symmetric key.

5.5.3. Error Response

The error responses for CoAP-based interactions with the AS are equivalent to the ones for HTTP-based interactions as defined in section 5.2 of [RFC6749], with the following differences:

- The Content-Type MUST be specified by the communication security profile used between client and AS. The raw payload before being processed by the communication security protocol MUST be encoded as a CBOR map.
- The CoAP response code 4.00 (Bad Request) MUST be used for all error responses, except for invalid_client where the CoAP response code 4.01 (Unauthorized) MAY be used under the same conditions as specified in section 5.2 of [RFC6749].
- The parameters "error", "error_description" and "error_uri" MAY be abbreviated using the codes specified in table Figure 13.
- The error codes MAY be abbreviated using the codes specified in table Figure 7.

error code	CBOR Key	Major Type
invalid_request	0	0 (uint)
invalid_client	1	0
invalid_grant	2	0
unauthorized_client	3	0
unsupported_grant_type	4	0
invalid_scope	5	0

Figure 7: CBOR abbreviations for common error codes

5.5.4. Request and Response Parameters

This section provides more detail about the new parameters that can be used in access token requests and responses, as well as abbreviations for more compact encoding of existing parameters and common parameter values.

5.5.4.1. Audience

This parameter specifies for which audience the client is requesting a token. It should be encoded as CBOR text string (major type 3). The formatting and semantics of these strings are application specific.

5.5.4.2. Grant Type

The abbreviations in Figure 8 MAY be used in CBOR encodings instead of the string values defined in [RFC6749].

grant_type	CBOR Key	Major Type
password	0	0 (uint)
authorization_code	1	0
client_credentials	2	0
refresh_token	3	0

Figure 8: CBOR abbreviations for common grant types

5.5.4.3. Token Type

The token_type parameter is defined in [RFC6749], allowing the AS to indicate to the client which type of access token it is receiving (e.g. a bearer token).

This document registers the new value "pop" for the OAuth Access Token Types registry, specifying a Proof-of-Possession token. How the proof-of-possession is performed MUST be specified by the profiles.

The values in the 'token_type' parameter MUST be CBOR text strings (major type 3).

In this framework token type 'pop' MUST be assumed by default if the AS does not provide a different value.

5.5.4.4. Profile

Profiles of this framework MUST define the communication protocol and the communication security protocol between the client and the RS. Furthermore profiles MUST define proof-of-possession methods, if they support proof-of-possession tokens.

A profile MUST specify an identifier that is used to uniquely identify itself in the 'profile' parameter.

Profiles MAY define additional parameters for both the token request and the RS Information in the access token response in order to support negotiation or signalling of profile specific parameters.

5.5.4.5. Confirmation

The "cnf" parameter identifies or provides the key used for proof-of-possession or for authenticating the RS depending on the proof-of-possession algorithm and the context cnf is used in. This framework extends the definition of 'cnf' from [RFC7800] by adding CBOR/COSE encodings and the use of 'cnf' for transporting keys in the RS Information.

The "cnf" parameter is used in the following contexts with the following meaning:

- In the access token, to indicate the proof-of-possession key bound to this token.
- In the token request C -> AS, to indicate the client's raw public key, or the key-identifier of a previously established key between C and RS.
- In the token response AS -> C, to indicate either the symmetric key generated by the AS for proof-of-possession or the raw public key used by the RS to authenticate.
- In the introspection response AS -> RS, to indicate the proof-of-possession key bound to the introspected token.
- In the client token AS -> RS -> C, to indicate the proof-of-possession key bound to the access token.

A CBOR encoded payload MAY contain the 'cnf' parameter with the following contents:

COSE_Key In this case the 'cnf' parameter contains the proof-of-possession key to be used by the client. An example is shown in Figure 9.

```
"cnf" : {
  "COSE_Key" : {
    "kty" : "EC",
    "kid" : h'11',
    "crv" : "P-256",
    "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
    "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
  }
}
```

Figure 9: Confirmation parameter containing a public key

Note that the COSE_Key structure may contain an "alg" or "key_ops" parameter. If such parameters are present, a client MUST NOT use a key that is not compatible with the profile or proof-of-possession algorithm according to those parameters.

COSE_Encrypted In this case the 'cnf' parameter contains an encrypted symmetric key destined for the client. The client is assumed to be able to decrypt the ciphertext of this parameter. The parameter is encoded as COSE_Encrypted object wrapping a COSE_Key object. Figure 10 shows an example of this type of encoding.

```
"cnf" : {
  "COSE_Encrypted" : {
    993(
      [ h'a1010a' # protected header : {"alg" : "AES-CCM-16-64-128"}
        "iv" : b64'ifUvZaHFgJM7UmGnjA',  # unprotected header
       b64'WXThuZo6TMCaZZqi6ef/8WHTjOdGk8kNzaIhIQ' # ciphertext
      ]
    )
  }
}
```

Figure 10: Confirmation parameter containing an encrypted symmetric key

The ciphertext here could e.g. contain a symmetric key as in Figure 11.

```
{
  "kty" : "Symmetric",
  "kid" : b64'39Gqlw',
  "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
}
```

Figure 11: Example plaintext of an encrypted cnf parameter

Key Identifier In this case the 'cnf' parameter references a key that is assumed to be previously known by the recipient. This allows clients that perform repeated requests for an access token for the same audience but e.g. with different scopes to omit key transport in the access token, token request and token response. Figure 12 shows such an example.

```
"cnf" : {
  "kid" : b64'39Gqlw'
}
```

Figure 12: A Confirmation parameter with just a key identifier

This specification establishes the IANA "CWT Confirmation Methods" registry for these types of confirmation methods in Section 8.10 and

registers the methods defined by this specification. Other specifications can register other methods used for confirmation. The registry is meant to be analogous to the "JWT Confirmation Methods" registry defined by [RFC7800].

5.5.5. Mapping parameters to CBOR

All OAuth parameters in access token requests and responses are mapped to CBOR types as follows and are given an integer key value to save space.

Parameter name	CBOR Key	Major Type
aud	3	3
client_id	8	3 (text string)
client_secret	9	2 (byte string)
response_type	10	3
redirect_uri	11	3
scope	12	3
state	13	3
code	14	2
error	15	3
error_description	16	3
error_uri	17	3
grant_type	18	0
access_token	19	3
token_type	20	0
expires_in	21	0
username	22	3
password	23	3
refresh_token	24	3
cnf	25	5 (map)
profile	26	3

Figure 13: CBOR mappings used in token requests

5.6. The 'Introspect' Endpoint

Token introspection [RFC7662] is used by the RS and potentially the client to query the AS for metadata about a given token e.g. validity or scope. Analogous to the protocol defined in RFC 7662 [RFC7662] for HTTP and JSON, this section defines adaptations to more constrained environments using CoAP and CBOR.

Communication between the RS and the introspection endpoint at the AS MUST be integrity protected and encrypted. Furthermore AS and RS

MUST perform mutual authentication. Finally the AS SHOULD verify that the RS has the right to access introspection information about the provided token. Profiles of this framework that support introspection MUST specify how authentication and communication security between RS and AS is implemented.

The figures of this section uses CBOR diagnostic notation without the integer abbreviations for the parameters or their values for better readability.

5.6.1. RS-to-AS Request

The RS sends a CoAP POST request to the introspection endpoint at the AS, the profile MUST specify the Content-Type and wrapping of the payload. The payload MUST be encoded as a CBOR map with a 'token' parameter containing the access token along with optional parameters representing additional context that is known by the RS to aid the AS in its response.

The same parameters are required and optional as in section 2.1 of RFC 7662 [RFC7662].

For example, Figure 14 shows a RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that we assume a object security-based communication security profile for this example, therefore the Content-Type is "application/cose+cbor".

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "introspect"
Content-Type: "application/cose+cbor"
Payload:
{
  "token" : b64'7gj0dXJQ43U',
  "token_type_hint" : "pop"
}
```

Figure 14: Example introspection request.

5.6.2. AS-to-RS Response

If the introspection request is authorized and successfully processed, the AS sends a response with the CoAP response code 2.01 (Created). If the introspection request was invalid, not authorized or couldn't be processed the AS returns an error response as described in Section 5.6.3.

In a successful response, the AS encodes the response parameters in a CBOR map including with the same required and optional parameters as in section 2.2. of RFC 7662 [RFC7662] with the following additions:

cnf
: OPTIONAL. This field contains information about the proof-of-possession key that binds the client to the access token. See Section 5.5.4.5 for more details on the formatting of the 'cnf' parameter.

profile
: OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.5.4.4 for more details on the formatting of this parameter.

client_token
: OPTIONAL. This parameter contains information that the RS MUST pass on to the client. See Section 5.6.4 for more details.

For example, Figure 15 shows an AS response to the introspection request in Figure 14. Note that we assume a DTLS-based communication security profile for this example, therefore the Content-Type is "application/cbor".

```
Header: Created Code=2.01)
Content-Type: "application/cbor"
Payload:
{
  "active" : true,
  "scope" : "read",
  "profile" : "coap_dtls",
  "client_token" : b64'2QPhg0OhAQo ...
  (remainder of client token omitted for brevity)',
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 15: Example introspection response.

5.6.3. Error Response

The error responses for CoAP-based interactions with the AS are equivalent to the ones for HTTP-based interactions as defined in section 2.3 of [RFC7662], with the following differences:

- If content is sent, the Content-Type MUST be set according to the specification of the communication security profile, and the content payload MUST be encoded as a CBOR map.
- If the credentials used by the RS are invalid the AS MUST respond with the CoAP response code 4.01 (Unauthorized) and use the required and optional parameters from section 5.2 in RFC 6749 [RFC6749].
- If the RS does not have the right to perform this introspection request, the AS MUST respond with the CoAP response code 4.03 (Forbidden). In this case no payload is returned.
- The parameters "error", "error_description" and "error_uri" MAY be abbreviated using the codes specified in table Figure 13.
- The error codes MAY be abbreviated using the codes specified in table Figure 7.

Note that a properly formed and authorized query for an inactive or otherwise invalid token does not warrant an error response by this specification. In these cases, the authorization server MUST instead respond with an introspection response with the "active" field set to "false".

5.6.4. Client Token

EDITORIAL NOTE: We have tentatively introduced this concept and would specifically like feedback whether this is viewed as a useful addition to the framework.

In cases where the client has limited connectivity and needs to get access to a previously unknown resource servers, this framework suggests the following approach: The client is pre-configured with a generic, long-term access token when it is commissioned. When the client then tries to access a RS it transmits this access token. The RS then performs token introspection to learn what access this token grants. In the introspection response, the AS also relays information for the client, such as the proof-of-possession key, through the RS. The RS passes on this Client Token to the client in response to the submission of the token.

The client_token parameter is designed to carry such information, and is intended to be used as described in Figure 16.

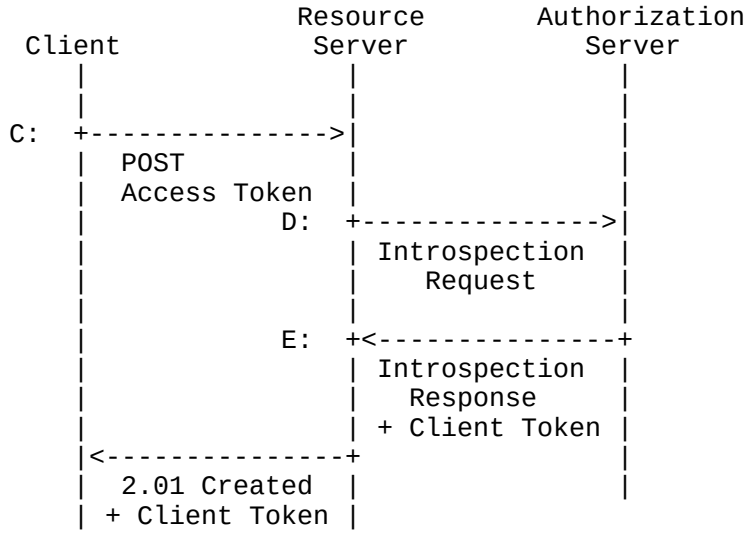


Figure 16: Use of the client_token parameter.

The client token is a COSE_Encrypted object, containing as payload a CBOR map with the following claims:

cnf
: REQUIRED if the token type is 'pop', OPTIONAL otherwise. Contains information about the proof-of-possession key the client is to use with its access token. See Section 5.5.4.5.

token_type
: OPTIONAL. See Section 5.5.4.3.

profile
: REQUIRED. See Section 5.5.4.4.

rs_cnf
: OPTIONAL. Contains information about the key that the RS uses to authenticate towards the client. If the key is symmetric then this claim MUST NOT be part of the Client Token, since this is the same key as the one specified through the 'cnf' claim. This claim uses the same encoding as the 'cnf' parameter. See Section 5.5.4.4.

The AS encrypts this token using a key shared between the AS and the client, so that only the client can decrypt it and access its payload. How this key is established is out of scope of this framework.

5.6.5. Mapping Introspection parameters to CBOR

The introspection request and response parameters are mapped to CBOR types as follows and are given an integer key value to save space.

Parameter name	CBOR Key	Major Type
iss	1	3 (text string)
sub	2	3
aud	3	3
exp	4	6 tag value 1
nbf	5	6 tag value 1
iat	6	6 tag value 1
cti	7	2 (byte string)
client_id	8	3
scope	12	3
token_type	20	3
username	22	3
cnf	25	5 (map)
profile	26	0 (uint)
token	27	3
token_type_hint	28	3
active	29	0
client_token	30	3
rs_cnf	31	5

Figure 17: CBOR Mappings to Token Introspection Parameters.

5.7. The Access Token

This framework RECOMMENDS the use of CBOR web token (CWT) as specified in [I-D.ietf-ace-cbor-web-token].

In order to facilitate offline processing of access tokens, this draft specifies the "cnf" and "scope" claims for CBOR web tokens.

The "scope" claim explicitly encodes the scope of a given access token. This claim follows the same encoding rules as defined in section 3.3 of [RFC6749]. The meaning of a specific scope value is application specific and expected to be known to the RS running that application.

The "cnf" claim follows the same rules as specified for JSON web token in RFC7800 [RFC7800], except that it is encoded in CBOR in the same way as specified for the "cnf" parameter in Section 5.5.4.5.

5.7.1. The 'Authorization Information' Endpoint

The access token, containing authorization information and information about the key used by the client, needs to be transported to the RS so that the RS can authenticate and authorize the client request.

This section defines a method for transporting the access token to the RS using CoAP. Profiles of this framework MAY define other methods for token transport.

The method consists of an /authz-info endpoint, implemented by the RS. A client using this method MUST make a POST request to /authz-info at the RS with the access token in the payload. The RS receiving the token MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with 2.01 (Created). This response MAY contain the identifier of the token (e.g. the cti for a CWT) as a payload.

If the token is not valid, the RS MUST respond with the CoAP response code 4.01 (Unauthorized). If the token is valid but the audience of the token does not match the RS, the RS MUST respond with the CoAP response code 4.03 (Forbidden). If the token is valid but is associated to claims that the RS cannot process (e.g. an unknown scope) the RS MUST respond with the CoAP response code 4.00 (Bad Request). In the latter case the RS MAY provide additional information in the error response, in order to clarify what went wrong.

The RS MAY make an introspection request to validate the token before responding to the POST /authz-info request. If the introspection response contains a client token (Section 5.6.4) then this token SHALL be included in the payload of the 2.01 (Created) response.

Profiles MUST specify how the /authz-info endpoint is protected. Note that since the token contains information that allow the client and the RS to establish a security context in the first place, mutual authentication may not be possible at this point.

The RS MUST be prepared to store more than one token for each client, and MUST apply the combined permissions granted by all applicable, valid tokens to client requests.

5.7.2. Token Expiration

Depending on the capabilities of the RS, there are various ways in which it can verify the validity of a received access token. We list

the possibilities here including what functionality they require of the RS.

- The token is a CWT/JWT and includes a 'exp' claim and possibly the 'nbf' claim. The RS verifies these by comparing them to values from its internal clock as defined in [RFC7519]. In this case the RS's internal clock must reflect the current date and time, or at least be synchronized with the AS's clock. How this clock synchronization would be performed is out of scope for this memo.
- The RS verifies the validity of the token by performing an introspection request as specified in Section 5.6. This requires the RS to have a reliable network connection to the AS and to be able to handle two secure sessions in parallel (C to RS and AS to RS).
- The RS and the AS both store a sequence number linked to their common security association. The AS increments this number for each access token it issues and includes it in the access token, which is a CWT. The RS keeps track of the most recently received sequence number, and only accepts tokens as valid, that are in a certain range around this number. This method does only require the RS to keep track of the sequence number. The method does not provide timely expiration, but it makes sure that older tokens cease to be valid after a certain number of newer ones got issued. For a constrained RS with no network connectivity and no means of reliably measuring time, this is the best that can be achieved.

If a token, that authorizes a long running request such as e.g. a CoAP Observe [RFC7641], expires, the RS MUST send an error response with the response code 4.01 Unauthorized to the client and then terminate processing the long running request.

6. 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.

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest. Consequently, the token integrity protection MUST be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key, this symmetric key MUST be encrypted by the authorization server with a long-term key shared with the resource server.

It is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. Using a single shared secret with multiple resource servers to simplify key management is NOT RECOMMENDED since the benefit from using the proof-of-possession concept is significantly reduced.

Token replay is also more difficult since an eavesdropper will have to obtain the token and the corresponding private key or shared secret that is bound to the access token. Nevertheless, it is good practice to limit the lifetime of the access token and therefore the lifetime of associated key.

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client will obtain the session key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby completely negating proof-of-possession security. Profiles MUST specify how confidentiality protection is provided, and additional protection can be applied by encrypting the token, for example encryption of CWTs is specified in section 5.1 of [I-D.ietf-ace-cbor-web-token].

Developers MUST ensure that the ephemeral credentials (i.e., the private key or the session key) are not leaked to third parties. An adversary in possession of the ephemeral credentials bound to the access token will be able to impersonate the client. Be aware that this is a real risk with many constrained environments, since adversaries can often easily get physical access to the devices.

Clients can at any time request a new proof-of-possession capable access token. Using a refresh token to regularly request new access tokens that are bound to fresh and unique keys is important if the client has this capability. Keeping the lifetime of the access token short allows the authorization server to use shorter key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens, they SHOULD scope these access tokens to a specific permissions. Furthermore access tokens SHOULD NOT apply to an audience that comprises more than one RS, since otherwise any RS in the audience can impersonate the client towards the other members of the audience.

Clients using an asymmetric key pair for proof-of-possession towards several different RS should be aware that they will need to rotate

that key pair more frequently than if it was only used towards a single RS.

7. Privacy Considerations

Implementers and users should be aware of the privacy implications of the different possible deployments of this framework.

The AS is in a very central position can potentially learn sensitive information about the clients requesting access tokens. If the client credentials grant is used, the AS can track what kind of access the client intends to perform. With other grants, the Resource Owner can bind the grants to anonymous (rotating) credentials, that do not allow the AS to link different access token requests by the same client.

If access tokens are only integrity protected and not encrypted, they may reveal information to attackers listening on the wire, or able to acquire the access tokens in some other way. In the case of CWTs or JWTs the token may e.g. reveal the audience, the scope and the confirmation method used by the client. The latter may reveal the client's identity.

Clients using asymmetric keys for proof-of-possession should be aware of the consequences of using the same key pair for proof-of-possession towards different RS. A set of colluding RS or an attacker able to obtain the access tokens will be able to link the requests, or even to determine the client's identity.

8. IANA Considerations

This specification registers new parameters for OAuth and establishes registries for mappings to CBOR.

8.1. OAuth Introspection Response Parameter Registration

This specification registers the following parameters in the OAuth introspection response parameters

- Name: "cnf"
- Description: Key to prove the right to use an access token, as defined in [RFC7800].
- Change Controller: IESG
- Specification Document(s): this document

- Name: "aud"
- Description: Reference to intended receiving RS, as defined in PoP token specification.

- Change Controller: IESG
- Specification Document(s): this document

- Name: "profile"
- Description: The communication and communication security profile used between client and RS, as defined in ACE profiles.
- Change Controller: IESG
- Specification Document(s): this document

- Name: "client_token"
- Description: Information that the RS MUST pass to the client e.g. about the proof-of-possession keys.
- Change Controller: IESG
- Specification Document(s): this document

- Name: "rs_cnf"
- Description: Describes the public key the RS uses to authenticate.
- Change Controller: IESG
- Specification Document(s): this document

8.2. OAuth Parameter Registration

This specification registers the following parameters in the OAuth Parameters Registry

- Parameter name: "profile"
- Parameter usage location: token request, and token response
- Change Controller: IESG
- Specification Document(s): this document

- Name: "cnf"
- Description: Key to prove the right to use an access token, as defined in [RFC7800].
- Change Controller: IESG
- Specification Document(s): this document

8.3. OAuth Access Token Types

This specification registers the following new token type in the OAuth Access Token Types Registry

- Name: "PoP"
- Description: A proof-of-possession token.
- Change Controller: IESG
- Specification Document(s): this document

8.4. Token Type Mappings

A new registry will be requested from IANA, entitled "Token Type Mappings". The registry is to be created as Expert Review Required.

8.4.1. Registration Template

Token Type:
: Name of token type as registered in the OAuth token type registry e.g. "Bearer".

Mapped value:
: Integer representation for the token type value. The key value MUST be an integer in the range of 1 to 65536.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter,preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.4.2. Initial Registry Contents

- Parameter name: "Bearer"
- Mapped value: 1
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "pop"
- Mapped value: 2
- Change Controller: IESG
- Specification Document(s): this document

8.5. CBOR Web Token Claims

This specification registers the following new claims in the CBOR Web Token (CWT) registry:

- Claim Name: "scope"
- Claim Description: The scope of an access token as defined in [RFC6749].
- Change Controller: IESG
- Specification Document(s): this document

- Claim Name: "cnf"

- Claim Description: The proof-of-possession key of an access token as defined in [RFC7800].
- Change Controller: IESG
- Specification Document(s): this document

8.6. ACE Profile Registry

A new registry will be requested from IANA, entitled "ACE Profile Registry". The registry is to be created as Expert Review Required.

8.6.1. Registration Template

Profile name:
: Name of the profile to be included in the profile attribute.

Profile description:
: Text giving an overview of the profile and the context it is developed for.

Profile ID:
: Integer value to identify the profile. The value MUST be an integer in the range of 1 to 65536.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter,preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.7. OAuth Parameter Mappings Registry

A new registry will be requested from IANA, entitled "Token Endpoint CBOR Mappings Registry". The registry is to be created as Expert Review Required.

8.7.1. Registration Template

Parameter name:
: OAuth Parameter name, refers to the name in the OAuth parameter registry e.g. "client_id".

CBOR key value:
: Key value for the claim. The key value MUST be an integer in the range of 1 to 65536.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
Reference to the document or documents that specify the parameter,preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.7.2. Initial Registry Contents

- Parameter name: "aud"
- CBOR key value: 3
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "client_id"
- CBOR key value: 8
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "client_secret"
- CBOR key value: 9
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "response_type"
- CBOR key value: 10
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "redirect_uri"
- CBOR key value: 11
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "scope"
- CBOR key value: 12
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "state"
- CBOR key value: 13
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "code"
- CBOR key value: 14
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "error"
- CBOR key value: 15
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "error_description"
- CBOR key value: 16
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "error_uri"
- CBOR key value: 17
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "grant_type"
- CBOR key value: 18
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "access_token"
- CBOR key value: 19
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "token_type"
- CBOR key value: 20
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "expires_in"
- CBOR key value: 21
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "username"
- CBOR key value: 22
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "password"
- CBOR key value: 23
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "refresh_token"
- CBOR key value: 24
- Change Controller: IESG

- Specification Document(s): this document

- Parameter name: "cnf"
- CBOR key value: 25
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "profile"
- CBOR key value: 26
- Change Controller: IESG
- Specification Document(s): this document

8.8. Introspection Endpoint CBOR Mappings Registry

A new registry will be requested from IANA, entitled "Introspection Endpoint CBOR Mappings Registry". The registry is to be created as Expert Review Required.

8.8.1. Registration Template

Response parameter name:
: Name of the response parameter as defined in the "OAuth Token Introspection Response" registry e.g. "active".

CBOR key value:
: Key value for the claim. The key value MUST be an integer in the range of 1 to 65536.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter,preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.8.2. Initial Registry Contents

- Response parameter name: "iss"
- CBOR key value: 1
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "sub"
- CBOR key value: 2
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "aud"
- CBOR key value: 3
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "exp"
- CBOR key value: 4
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "nbf"
- CBOR key value: 5
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "iat"
- CBOR key value: 6
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "cti"
- CBOR key value: 7
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "client_id"
- CBOR key value: 8
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "scope"
- CBOR key value: 12
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "token_type"
- CBOR key value: 20
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "username"
- CBOR key value: 22
- Change Controller: IESG
- Specification Document(s): this document

- Parameter name: "cnf"
- CBOR key value: 25
- Change Controller: IESG

- Specification Document(s): this document

- Parameter name: "profile"
- CBOR key value: 26
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "token"
- CBOR key value: 27
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "token_type_hint"
- CBOR key value: 28
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "active"
- CBOR key value: 29
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "client_token"
- CBOR key value: 30
- Change Controller: IESG
- Specification Document(s): this document

- Response parameter name: "rs_cnf"
- CBOR key value: 31
- Change Controller: IESG
- Specification Document(s): this document

8.9. CoAP Option Number Registration

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

Name

Access-Token

Number

TBD

Reference

[This document].

Meaning in Request

Contains an Access Token according to [This document] containing access permissions of the client.

Meaning in Response

Not used in response

Safe-to-Forward

Yes

Format

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

Length

Less then 255 bytes

8.10. CWT Confirmation Methods Registry

This specification establishes the IANA "CWT Confirmation Methods" registry for CWT "cnf" member values. The registry records the confirmation method member and a reference to the specification that defines it.

8.10.1. Registration Template

Confirmation Method Name:
The name requested (e.g., "kid"). This name is intended to be human readable and be used for debugging purposes. It is case sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Experts state that there is a compelling reason to allow an exception.

Confirmation Method Value:
Integer representation for the confirmation method value. Intended for use to uniquely identify the confirmation method. The value MUST be an integer in the range of 1 to 65536.

Confirmation Method Description:
Brief description of the confirmation method (e.g. "Key Identifier").

Change Controller:
For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):

Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

8.10.2. Initial Registry Contents

- Confirmation Method Name: "COSE_Key"
- Confirmation Method Value: 1
- Confirmation Method Description: A COSE_Key that is either a public key or a symmetric key.
- Change Controller: IESG
- Specification Document(s): this document

- Confirmation Method Name: "COSE_Encrypted"
- Confirmation Method Value: 2
- Confirmation Method Description: A COSE_Encrypted structure that wraps a COSE_Key containing a symmetric key.
- Change Controller: IESG
- Specification Document(s): this document

- Confirmation Method Name: "Key Identifier"
- Confirmation Method Value: 3
- Confirmation Method Description: A key identifier.
- Change Controller: IESG
- Specification Document(s): this document

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 predecessors of this proposal. Finally, we would like to thank the ACE working group in general for their feedback.

We would like to thank the authors of draft-ietf-oauth-pop-key-distribution, from where we copied large parts of our security considerations.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

10. References

10.1. Normative References

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

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

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

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

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <http://www.rfc-editor.org/info/rfc7662>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <http://www.rfc-editor.org/info/rfc7800>.

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-05 (work in progress), March 2017.

[I-D.ietf-ace-cbor-web-token]
Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-03 (work in progress), March 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.ietf-oauth-device-flow]
Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Flow for Browserless and Input Constrained Devices", draft-ietf-oauth-device-flow-04 (work in progress), February 2017.

[I-D.ietf-oauth-native-apps]
Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", draft-ietf-oauth-native-apps-09 (work in progress), March 2017.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <http://www.rfc-editor.org/info/rfc6819>.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <http://www.rfc-editor.org/info/rfc7521>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <http://www.rfc-editor.org/info/rfc7591>.

[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>.

[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, <http://www.rfc-editor.org/info/rfc7744>.

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

Appendix A. Design Justification

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

Common IoT constraints are:

Low Power Radio:

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

Low CPU Speed:

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

Small Amount of Memory:

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

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers do not want to give access to the data collected by their IoT device or to functions it may offer to third parties. Since the classical approach of requesting permissions from end users via a

rich user interface does not work in many IoT deployment scenarios these functions need to be delegated to user controlled devices that are better suitable for such tasks, such as smart phones and tablets.

Communication Constraints:

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

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

Appendix B. Roles and Responsibilities

Resource Owner

* Make sure that the RS is registered at the AS. This includes making known to the AS which profiles, token_types, scopes, and key types (symmetric/asymmetric) the RS supports. Also making it known to the AS which audience(s) the RS identifies itself with.
* Make sure that clients can discover the AS which is in charge of the RS.
* If the client-credentials grant is used, make sure that the AS has the necessary, up-to-date, access control policies for the RS.

Requesting Party

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

* Register the client at the AS. This includes making known to the AS which profiles, token_types, and key types (symmetric/ asymmetric) the client.

Authorization Server

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

Client

* Discover the AS in charge of the RS that is to be targeted with a request.
* Submit the token request (A).
  + Authenticate towards the AS.
  + Optionally (if not pre-configured): Specify which RS, which resource(s), and which action(s) the request(s) will target.
  + If raw public key (rpk) or certificate is used, make sure the AS has the right rpk or certificate for this client.
* Process the access token and RS Information (B)
  + Check that the RS Information provides the necessary security parameters (e.g. PoP key, information on communication security protocols supported by the RS).
* Send the token and request to the RS (C)
  + Authenticate towards the RS (this could coincide with the proof of possession process).
  + Transmit the token as specified by the AS (default is to the /authz-info endpoint, alternative options are specified by profiles).
  + Perform the proof-of-possession procedure as specified by the profile in use (this may already have been taken care of through the authentication procedure).

* Process the RS response (F) requirements of the Requesting Party, when issuing requests (e.g. minimum communication security requirements, trust anchors).
* Register the client at the AS.

Resource Server

* Expose a way to submit access tokens. By default this is the /authz-info endpoint.
* Process an access token.
  + Verify the token is from the right AS.
  + Verify that the token applies to this RS.
  + Check that the token has not expired (if the token provides expiration information).
  + Check the token's integrity.
  + Store the token so that it can be retrieved in the context of a matching request.
* Process a request.
  + Set up communication security with the client.
  + Authenticate the client.
  + Match the client against existing tokens.
  + Check that tokens belonging to the client actually authorize the requested action.
  + Optionally: Check that the matching tokens are still valid, using introspection (if this is possible.)
* Send a response following the agreed upon communication security.

Appendix C. Requirements on Profiles

This section lists the requirements on profiles of this framework, for the convenience of a profile designer.

- o Optionally Specify the discovery process of how the client finds the right AS for an RS it wants to send a request to. Section 4
- o Specify the communication protocol the client and RS the must use (e.g. CoAP). Section 5 and Section 5.5.4.4
- o Specify the security protocol the client and RS must use to protect their communication (e.g. OSCOAP or DTLS over CoAP). This must provide encryption and integrity protection. Section 5.5.4.4
- o Specify how the client and the RS mutually authenticate. Section 4
- o Specify the Content-format of the protocol messages (e.g. "application/cbor" or "application/cose+cbor"). Section 4

- Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g. symmetric/asymmetric) are supported by a specific proof-of-possession protocol. Section 5.5.4.3
- Specify a unique profile identifier. Section 5.5.4.4
- Optionally specify how the RS talks to the AS for introspection.Section 5.6
- Optionally specify how the client talks to the AS for requesting a token. Section 5.5
- Specify how/if the /authz-info endpoint is protected. Section 5.7.1
- Optionally define other methods of token transport than the /authz-info endpoint. Section 5.7.1

Appendix D. Assumptions on AS knowledge about C and RS

This section lists the assumptions on what an AS should know about a client and a RS in order to be able to respond to requests to the /token and /introspect endpoints. How this information is established is out of scope for this document.

- The identifier of the client or RS.
- The profiles that the client or RS supports.
- The scopes that the RS supports.
- The audiences that the RS identifies with.
- The key types (e.g. pre-shared symmetric key, raw public key, key length, other key parameters) that the client or RS supports.
- The types of access tokens the RS supports (e.g. CWT).
- If the RS supports CWTs, the COSE parameters for the crypto wrapper (e.g. algorithm, key-wrap algorithm, key-length).
- The expiration time for access tokens issued to this RS (unless the RS accepts a default time chosen by the AS).
- The symmetric key shared between client or RS and AS (if any).
- The raw public key of the client or RS (if any).

Appendix E. Deployment Examples

There is a large variety of IoT deployments, as is indicated in Appendix A, and this section highlights a few 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 the the security of the requests and responses between the clients and the RS to consider.

Note: CBOR diagnostic notation is used for examples of requests and responses.

E.1. Local Token Validation

In this scenario we consider the case where the resource server is offline, i.e. it is not connected to the AS at the time of the access request. This access procedure involves steps A, B, C, and F of Figure 1.

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 18.

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 the AS. The security of this request can be transport or application layer, it is up the the communication security profile to define. In the example transport layer identification of the AS is done and the client identifies with client_id and client_secret as in classic OAuth. The request contains the public key of the client and the Audience parameter set to "tempSensorInLivingRoom", a value that the temperature sensor identifies itself with. The AS evaluates the request and authorizes the client to access the resource.
B: The AS responds with a PoP token and RS Information. The PoP token contains the public key of the client, and the RS Information contains the public key of the RS. For communication security this example uses DTLS RawPublicKey between the client and the RS. The issued token will have a short validity time, i.e. 'exp' close to 'iat', to protect the RS from replay attacks. The token includes the claim such as "scope" with the authorized access that an owner of the temperature device can enjoy. In this example, the 'scope' claim, issued by the AS, informs the RS that the owner of the token, that can prove the possession of a key is authorized to make a GET request against the /temperature resource and a POST request on the /firmware resource. Note that the syntax and semantics of the scope claim are application specific.
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.

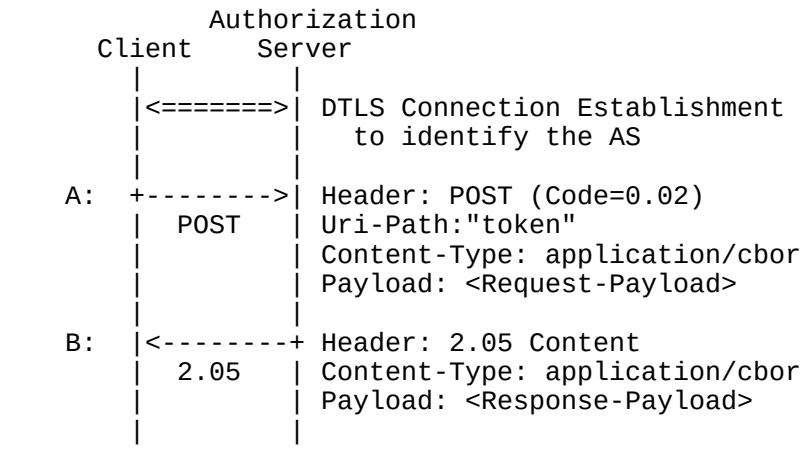


Figure 18: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 19. Note that we assume a DTLS-based communication security profile for this example, therefore the Content-Type is "application/cbor".

```
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",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x"   : b64'MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y"   : b64'4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
    }
  }
}
```

Figure 19: Request and Response Payload Details.

The content of the access token is shown in Figure 20.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "jwk" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}
```

Figure 20: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 21 - Figure 22.

C: The client then sends the PoP token to the /authz-info endpoint at the RS. This is a plain CoAP request, i.e. no transport or application layer security 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.

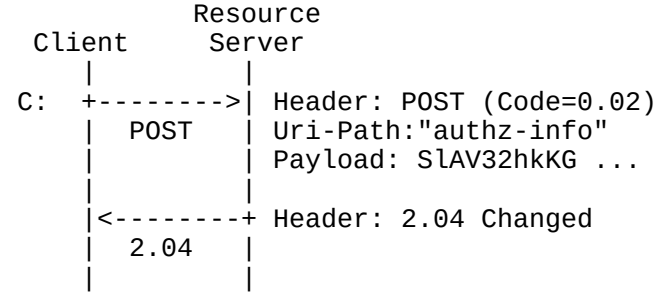


Figure 21: Access Token provisioning to RS

The client and the RS runs 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, based on previously established security context.
F: The RS responds with a resource representation over DTLS.

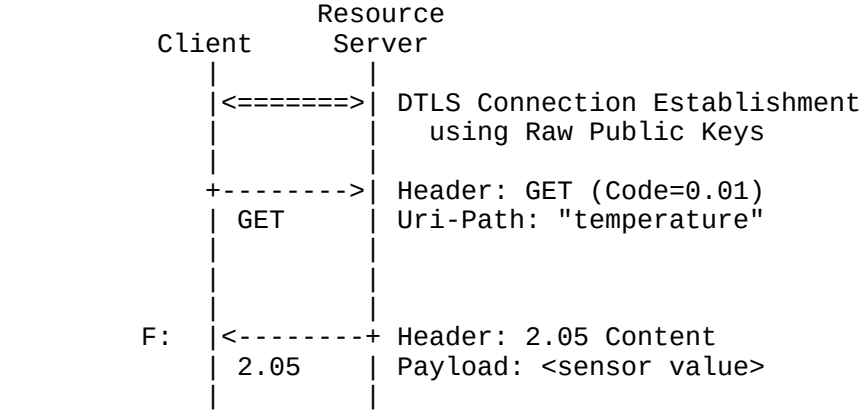


Figure 22: Resource Request and Response protected by DTLS.

E.2. Introspection Aided Token Validation

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

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

In the example interactions between an offline client (key fob), a RS (online lock), and an AS is shown. 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 23.

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 can used to authorize the key for his car at the the car manufacturers AS.

Note: In this example the client does not know the exact door it will be used to access since the token request is not send at the time of access. So the scope and audience parameters is set quite wide to start with and new values different form the original once can be returned from introspection later on.

A: The client sends the request using POST to /token at AS. The request contains the Audience parameter set to "PACS1337" (PACS, Physical Access System), 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. The security is provided by identifying the AS on transport layer using a pre shared security context (psk, rpk or certificate) and then the client is identified using client_id and client_secret as in classic OAuth
B: The AS responds with the an access token and RS Information, the latter containing a symmetric key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key being used as the PreSharedKey.

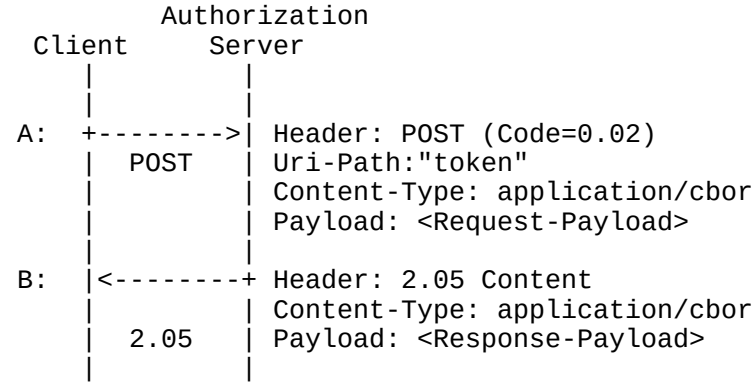


Figure 23: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 24.

```
Request-Payload:
{
  "grant_type" : "client_credentials",
  "aud" : "lockOfDoor4711",
  "client_id" : "keyfob",
  "client_secret" : "qwerty"
}

Response-Payload:
{
  "access_token" : b64'SlAV32hkKG ...'
  "token_type" : "pop",
  "csp" : "DTLS",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "oct",
      "alg" : "HS256",
      "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
    }
  }
}
```

Figure 24: 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 endpoint in the RS. This is a plain CoAP request, i.e. no DTLS 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 after step E.
D: The RS forwards the token to the /introspect endpoint on the AS. Introspection assumes a secure connection between the AS and the RS, e.g. using transport of application layer security. In the example AS is identified using pre shared security context (psk, rpk or certificate) while RS is acting as client and is identified with client_id and client_secret.
E: The AS provides the introspection response containing parameters about the token. This includes the confirmation key (cnf) parameter 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 the CoAP response code 2.01 (Created).

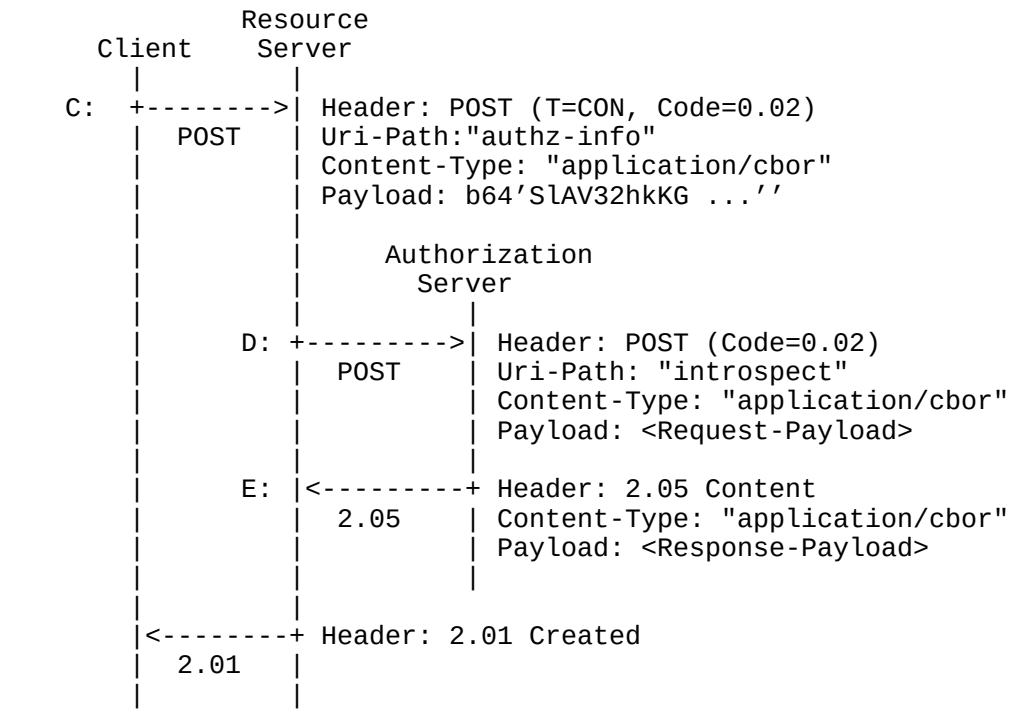


Figure 25: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 26.

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

Response-Payload:
{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1311280970,
  "cnf" : {
    "kid" : b64'JDLUhTMjU2IiwiY3R5Ijoi ...'
  }
}
```

Figure 26: Request and Response Payload for Introspection

The client uses the symmetric PoP key to establish a DTLS PreSharedKey secure connection to the RS. The CoAP request PUT is sent to the uri-path /state on RS changing state of the door to locked.
F: The RS responds with a appropriate over the secure DTLS channel.

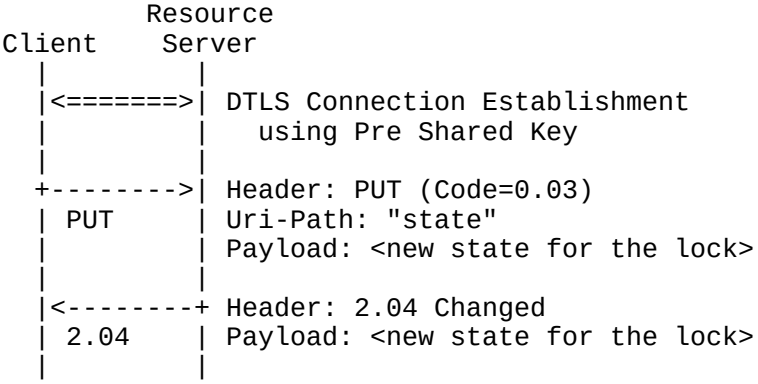


Figure 27: Resource request and response protected by OSCOAP

Appendix F. Document Updates

F.1. Version -05 to -06

- Moved sections that define the ACE framework into a subsection of the framework Section 5.
- Split section on client credentials and grant into two separate sections, Section 5.1, and Section 5.2.
- Added Section 5.3 on AS authentication.
- Added Section 5.4 on the Authorize endpoint.

F.2. Version -04 to -05

- Added RFC 2119 language to the specification of the required behavior of profile specifications.
- Added Section 5.2 on the relation to the OAuth2 grant types.
- Added CBOR abbreviations for error and the error codes defined in OAuth2.
- Added clarification about token expiration and long-running requests in Section 5.7.2
- Added security considerations about tokens with symmetric pop keys valid for more than one RS.
- Added privacy considerations section.
- Added IANA registry mapping the confirmation types from RFC 7800 to equivalent COSE types.

- Added appendix D, describing assumptions about what the AS knows about the client and the RS.

F.3. Version -03 to -04

- Added a description of the terms "framework" and "profiles" as used in this document.
- Clarified protection of access tokens in section 3.1.
- Clarified uses of the 'cnf' parameter in section 6.4.5.
- Clarified intended use of Client Token in section 7.4.

F.4. Version -02 to -03

- Removed references to draft-ietf-oauth-pop-key-distribution since the status of this draft is unclear.
- Copied and adapted security considerations from draft-ietf-oauth-pop-key-distribution.
- Renamed "client information" to "RS information" since it is information about the RS.
- Clarified the requirements on profiles of this framework.
- Clarified the token endpoint protocol and removed negotiation of 'profile' and 'alg' (section 6).
- Renumbered the abbreviations for claims and parameters to get a consistent numbering across different endpoints.
- Clarified the introspection endpoint.
- Renamed token, introspection and authz-info to 'endpoint' instead of 'resource' to mirror the OAuth 2.0 terminology.
- Updated the examples in the appendices.

F.5. Version -01 to -02

- Restructured to remove communication security parts. These shall now be defined in profiles.
- Restructured section 5 to create new sections on the OAuth endpoints /token, /introspect and /authz-info.
- Pulled in material from draft-ietf-oauth-pop-key-distribution in order to define proof-of-possession key distribution.
- Introduced the 'cnf' parameter as defined in RFC7800 to reference or transport keys used for proof of possession.
- Introduced the 'client-token' to transport client information from the AS to the client via the RS in conjunction with introspection.
- Expanded the IANA section to define parameters for token request, introspection and CWT claims.
- Moved deployment scenarios to the appendix as examples.

F.6. Version -00 to -01

- Changed 5.1. from "Communication Security Protocol" to "Client Information".
- Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP token request profile for IoT.
  - Allow the client to indicate preferences for the communication security protocol.
  - Defined the term "Client Information" for the additional information returned to the client in addition to the access token.
  - Require that the messages between AS and client are secured, either with (D)TLS or with COSE_Encrypted wrappers.
  - Removed dependency on OSCOAP and added generic text about object security instead.
  - Defined the "rpk" parameter in the client information to transmit the raw public key of the RS from AS to client.
  - (D)TLS MUST use the PoP key in the handshake (either as PSK or as client RPK with client authentication).
  - Defined the use of x5c, x5t and x5tS256 parameters when a client certificate is used for proof of possession.
  - Defined "tktn" parameter for signaling for how to transfer the access token.
- Added 5.2. the CoAP Access-Token option for transferring access tokens in messages that do not have payload.
- 5.3.2. Defined success and error responses from the RS when receiving an access token.
- 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
- Appendix B Added list of roles and responsibilities for C, AS and RS.

Authors' Addresses

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig@ri.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
SWEDEN

Email: goran.selander@ericsson.com

Erik Wahlstroem
(no affiliation)
Sweden

Email: erik@wahlstromtekniska.se

Samuel Erdtman
Spotify AB
Birger Jarlsgatan 61, 4tr
Stockholm 113 56
Sweden

Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

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

F. Palombini
Ericsson

March 13, 2017

CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE) draft-palombini-ace-coap-pubsub-profile-00

Abstract

This specification defines a profile for authentication and authorization for publishers and subscribers in a pub-sub setting scenario in a constrained environment, using the ACE framework. This profile relies on transport layer or application layer security to authorize the publisher to the broker. Moreover, it relies on application layer security for publisher-broker and subscriber-broker communication.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

This specification defines a way to authorize nodes in a CoAP pub-sub type of setting, using the ACE framework [I-D.ietf-ace-oauth-authz]. The pub-sub scenario is described in [I-D.ietf-core-coap-pubsub].

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 expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz] and [I-D.ietf-core-coap-pubsub].

2. Overview

The objective of this specification is to specify how to protect a CoAP pub-sub communication, as described in [I-D.ietf-core-coap-pubsub], using Ace framework ([I-D.ietf-ace-oauth-authz]) and profiles ([I-D.gerdes-ace-dtls-authorize], [I-D.seitz-ace-oscoap-profile]).

The architecture of the scenario is shown in Figure 1.

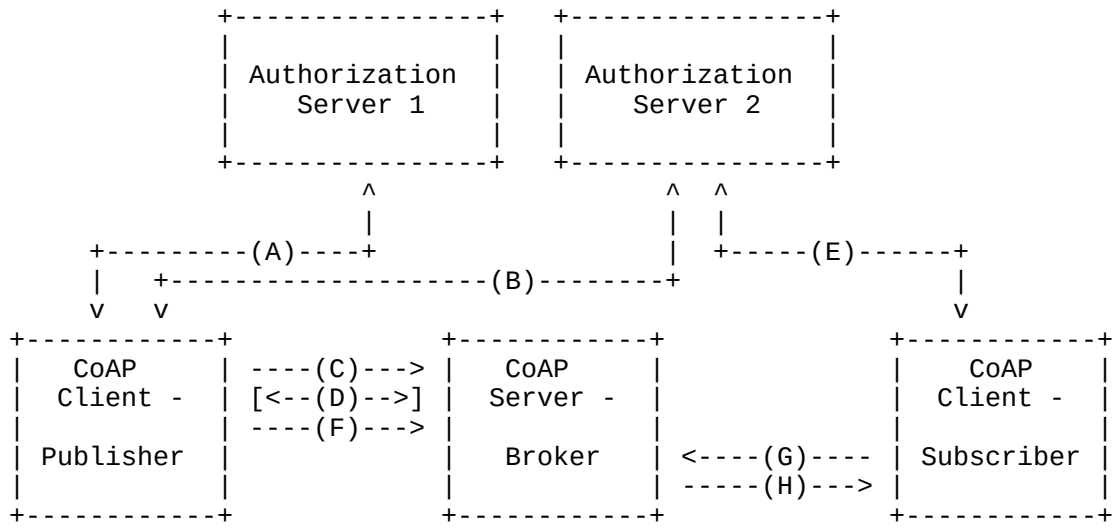


Figure 1: Architecture CoAP pubsub with Authorization Servers

The RS is the broker, which contains the topic. The AS1 hosts the policies about the Broker: what endpoints are allowed to Publish on the Broker. The AS2 hosts the policies about the topic: what endpoints are allowed to access what topic. There are four phases, the first three can be done in parallel.

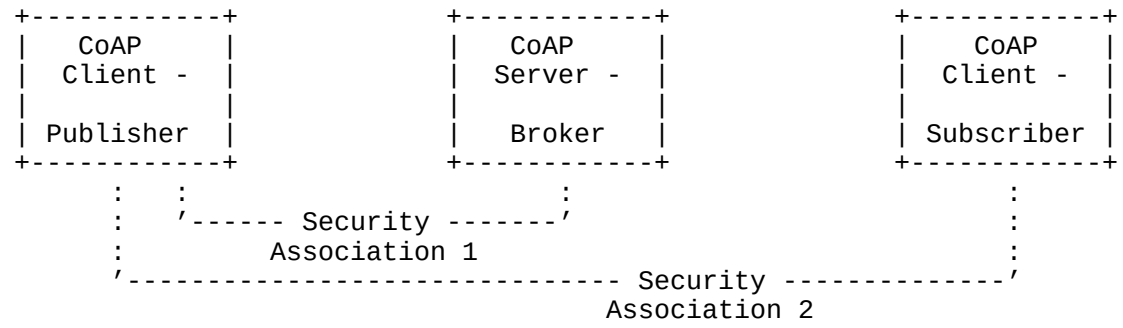
1. The Publisher requests publishing access to a broker at the AS1, and communicates with the Broker to set up security.

2. The Publisher requests access to a specific topic at the AS2

3. The Subscriber requests access to a specific topic at the AS2.

4. The Publisher and the Subscriber securely post to and get publications from the Broker.

This scenario requires the setup of 2 different security associations: on the one hand, the Publisher has a security association with the Broker, to protect the communication and securely authorize the Publisher to publish on a topic (Security Association 1). On the other hand, the Publisher has a security association with the Subscriber, to protect the publication content itself (Security Association 2). The Security Association 1 is set up using AS1, the Security Association 2 is set up using AS2.



3. Publisher Profile

In this section, it is specified how the Publisher requests, obtains and communicates to the Broker the access token, as well as the retrieval of the keying material to protect the publication.

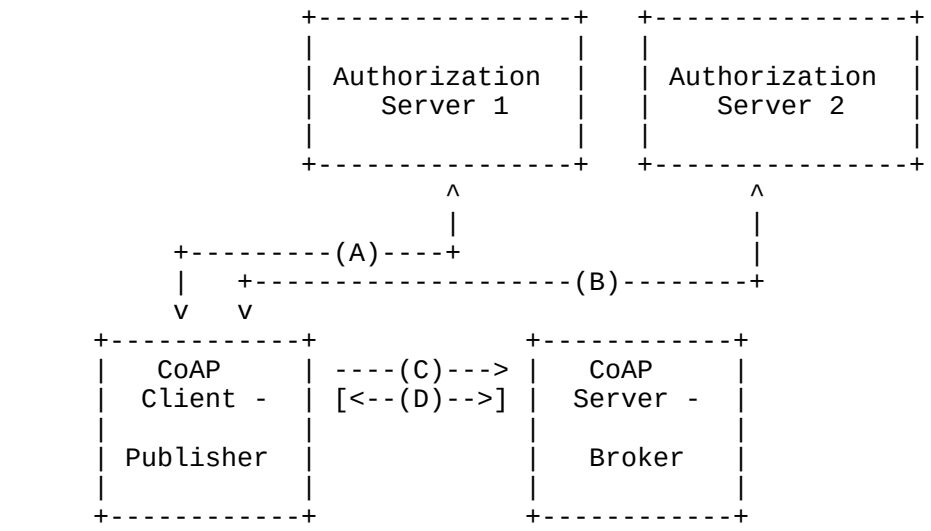


Figure 2: Phase 1: Publisher side

This is a combination of two independent phases:

- one is the establishment of a secure connection between Publisher and Broker, using an ACE profile such as DTLS [I-D.gerdes-ace-dtls-authorize] or OSCOAP [I-D.seitz-ace-oscoap-profile]. (A)(C)(D)

- the other is the Publisher's retrieval of keying material to protect the publication. (B)

In detail:

(A) corresponds to the Access Token Request and Response between Publisher and Authorization Server to retrieve the Access Token and RS (Broker) Information. As specified, the Publisher has the role of a CoAP client, the Broker has the role of the CoAP server.

(C) corresponds to the exchange between Publisher and Broker, where the Publisher sends its access token to the Broker.

(D) corresponds to the exchange where the Publisher establishes a secure connection with the Broker. Depending on the Information received in (A), this can be for example DTLS handshake, or other protocols such as EDHOC. Depending on the application, there may not be the need for this set up phase: for example, if OSCOAP is used directly and not without EDHOC first.

(A), (C) and (D) details are specified in the profile used.

(B) corresponds to the retrieval of the keying material to protect the publication. The detailed message flow is defined below.

3.1. Retrieval of COSE Key for protection of content

This phase is common to both Publisher and Subscriber. To maintain the generality, the Publisher or Subscriber is referred as Client in this section.

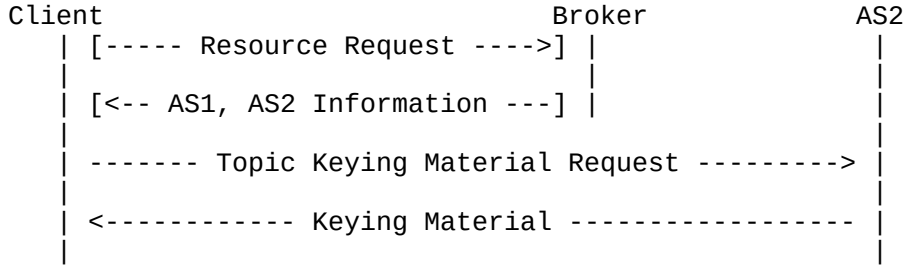


Figure 3: B: Access request - response

Complementary to what is defined in the DTLS profile (section 2.), to determine the AS2 in charge of a topic hosted at the broker, the Broker MAY send the address of both the AS in charge of the topic back to the Client, as a response to a Resource Request (Section 2.1). Analogously to the DTLS profile, instead of the initial Unauthorized Resource Request message, the Client MAY look up the desired topic in a resource directory (see [I-D.ietf-core-resource-directory]).

After retrieving the AS2 address, the Client sends a Topic Keying Material Request, which is a token-less authorization as described in [I-D.seitz-ace-oauth-authz], section 6.5. More specifically, the Client sends a POST request to the /token endpoint on AS2, that MUST contain in the payload:

- the grant type set to "client_credentials",
- the audience parameter set to the Broker,
- the scope parameter set to the topic,
- the cnf parameter containing the Client's COSE key, if the Client is a publisher, and
- OPTIONALLY, other additional parameters such as the client id or the algorithm.

Note that, if present, the algorithm MUST be a Content Encryption Algorithm, as defined in Section 10 of [I-D.ietf-cose-msg]. An example of the payload of a Topic Keying Material Request for a Publisher is specified in Figure 4.

```
{
  "grant_type" : "client_credentials",
  "aud" : "Broker1",
  "scope" : "Temp",
  "client_id" : "publisher1",
  "cnf" :
    { / COSE_Key /
      / type / 1 : 2, / EC2 /
      / kid / 2 : h'11',
      / alg / 3 : -7, / ECDSA with SHA-256 /
      / crv / -1 : 1 , / P-256 /
      / x / -2 : h'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de1
      08de439c08551d',
      / y /-3 : h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e
      9eecd0084d19c'
    }
}
```

Figure 4: Example of Topic Keying Material Request payload for a Publisher

The AS2 verifies that the Client is authorized to access the topic and, if the "cnf" parameter is present, stores the public key of the Client.

The AS2 response contains an empty token and the keying material to protect the publication ("key" field in the payload). Moreover, the payload MUST contain the "profile" parameter, set to value "OSCON", and the "token_type" set to "none".

TODO: define "key" parameter following ACE framework

The "key" parameter value MUST be a serialized COSE Key (see Section 7 of [I-D.ietf-cose-msg]), with the following values:

- kty with value 4 (symmetric)
- alg with value defined by the AS2 (Content Encryption Algorithm)
- k with value the symmetric key value
- OPTIONALLY, kid with an identifier for the key value

An example for the response is detailed in Figure 5.

```
{
  "access_token" : NULL,
  "token_type" : "none",
  "profile" : "OSCON",
  "key" : h'a4010402421234030c205002e2cc3a9b92855220f255fff1c615bc'
 /{1: 4, 2: h'1234', 3: 12, -1: h'02e2cc3a9b92855220f255fff1c615bc'}/
}
```

Figure 5: Example of Topic Keying Material response payload for a Publisher

3.2. AS1, AS2 Information

The Client MUST be able to process the following response message from the Broker, in order to retrieve the correct AS1 and AS2 addresses.

This CoAP message MUST have the following characteristics: the CoAP Code MUST be 4.01 "Unauthorized", the payload MUST be present and MUST include the full URI of both AS. An example using CBOR diagnostic notation is given below:

```
4.01 Unauthorized
Content-Format: application/ace+cbor
{"AS1": "coaps://as1.example.com/token",
 "AS2": "coaps://as2.example.com/pubsubkey"}
```

Figure 6: AS1, AS2 Information example

4. Subscriber Profile

In this section, it is specified how the Subscriber retrieves the keying material to protect the publication.

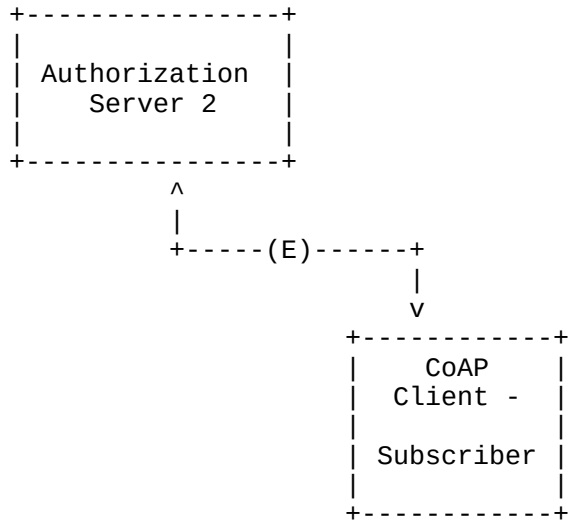


Figure 7: Phase 2: Subscriber side

Step (E) between Subscriber and AS2 corresponds to the retrieval of the keying material to verify the publication, and is the same as (B) between Publisher and AS2 (Section 3.1), with the following differences:

- The POST request to the /token endpoint on AS2, does not contain the cnf parameter containing the Client's COSE key.
- The AS2 response contains a "cnf" parameter whose value is set to a COSE Key Set, (Section 7 of [I-D.ietf-cose-msg]) i.e. an array of COSE Keys, which contains the public keys of all authorized Publishers

An example of the payload of a Topic Keying Material Request and corresponding response for a Subscriber is specified in Figure 8 and Figure 9.

```
{
  "grant_type" : "client_credentials",
  "aud" : "Broker1",
  "scope" : "Temp",
  "client_id" : "subscriber1"
}
```

Figure 8: Example of Topic Keying Material Request payload for a Subscriber

```
{
  "access_token" : NULL,
  "token_type" : "none",
  "profile" : "OSCON",
  "key" : h'a40104024212340 30c205002e2cc3a9b92855220f255fff1c615bc',
 /{1: 4, 2: h'1234', 3: 12, -1: h'02e2cc3a9b92855220f255fff1c615bc'}/
  "cnf" : [
   {
      1 : 2, / type EC2 /
      2 : h'11', / kid /
      3 : -7, / alg ECDSA with SHA-256 /
      -1 : 1 , / crv P-256 /
      -2 : h'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de43
      9c08551d', / x /
      -3 : h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd
      0084d19c' / y /
    }
  ]
}
```

Figure 9: Example of Topic Keying Material response payload for a Subscriber

5. Pub-Sub Protected Communication

This section specifies the communication Publisher-Broker and Subscriber-Broker, after the previous phases have taken place.

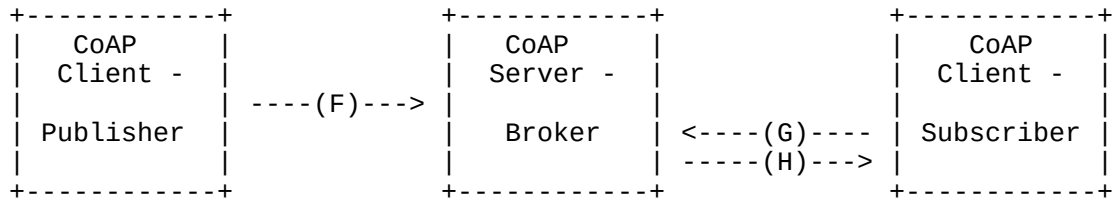


Figure 10: Phase 3: Secure communication between Publisher and Subscriber

The (F) message corresponds to the publication of a topic on the Broker. The publication (the resource representation) is protected with COSE ([I-D.ietf-cose-msg]). The (G) message is the subscription of the Subscriber, which is unprotected. The (H) message is the response from the Broker, where the publication is protected with COSE.

The flow graph is presented below.

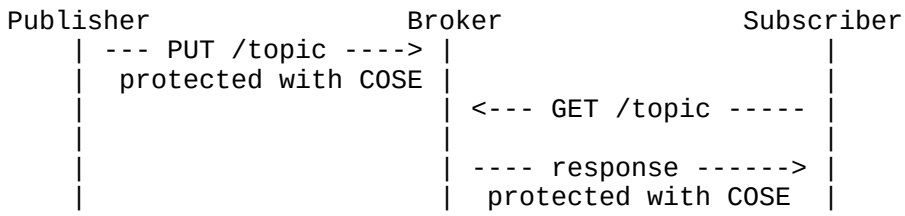


Figure 11: (F), (G), (H): Example of protected communication

5.1. Using COSE Objects to protect the resource representation

The Publisher uses the symmetric COSE Key received from AS2 in exchange B (Section 3.1) to protect the payload of the PUBLISH operation (Section 4.3 of [I-D.ietf-core-coap-pubsub]). Specifically, the COSE Key is used to create a COSE_Encrypt0 with algorithm specified by AS2. The Publisher uses the private key corresponding to the public key sent to the AS2 in exchange B (Section 3.1) to countersign the COSE Object as specified in Section 4.5 of [I-D.ietf-cose-msg]. The CoAP payload is replaced by the COSE object before the publication is sent to the Broker.

The Subscriber uses the kid in the countersignature field in the COSE object to retrieve the right public key to verify the countersignature. It then uses the symmetric key received from AS2 to verify and decrypt the publication received in the payload of the CoAP Notification from the Broker.

The COSE object is constructed in the following way:

- The protected Headers (as described in Section 3 of [I-D.ietf-cose-msg]) MAY contain the kid parameter, with value the kid of the symmetric COSE Key received in Section 3.1 and MUST contain the content encryption algorithm

- The unprotected Headers MUST contain the IV and the counter signature that includes:

* the algorithm (same value as in the asymmetric COSE Key received in (B)) in the protected header

* the kid (same value as the kid of the asymmetric COSE Key received in (B)) in the unprotected header

* the signature computed as specified in Section 4.5 of [I-D.ietf-cose-msg]

o The ciphertext, computed over the plaintext that MUST contain the CoAP payload.

The external_aad, when using AEAD, is an empty string.

An example is given in Figure 12

```
16(
  [
    / protected / h'a2010c04421234' / {
        \ alg \ 1:12, \ AES-CCM-64-64-128 \
        \ kid \ 4: h'1234'
      } / ,
    / unprotected / {
      / iv / 5:h'89f52f65a1c580',
      / countersign / 7:[
        / protected / h'a10126' / {
          \ alg \ 1:-7
        } / ,
        / unprotected / {
          / kid / 4:h'11'
        },
        / signature / SIG / 64 bytes signature /
      ]
    },
    / ciphertext / h'8df0a3b62fccff37aa313c8020e971f8aC8d'
  ]
)
```

Figure 12: Example of COSE Object sent in the payload of a PUBLISH operation

The encryption and decryption operations are described in sections 5.3 and 5.4 of [I-D.ietf-cose-msg].

6. Security Considerations

In the profile described above, the Publisher and Subscriber use asymmetric crypto, which would make the message exchange quite heavy for small constrained devices. Moreover, all Subscribers must be able to access the public keys of all the Publishers to a specific topic to be able to verify the publications. Such a database could be set up and managed by the same entity having control of the topic, i.e. AS2.

An application where it is not critical that only authorized Publishers can publish on a topic may decide not to make use of the asymmetric crypto and only use symmetric encryption/MAC to confidentiality and integrity protect the publication, but this is not recommended since, as a result, any authorized Subscribers with access to the Broker may forge unauthorized publications without being detected. In this symmetric case the Subscribers would only need one symmetric key per topic, and would not need to know any information about the Publishers, that can be anonymous to it and the Broker.

Subscribers can be excluded from future publications through re-keying for a certain topic. This could be set up to happen on a regular basis, for certain applications. How this could be done is out of scope for this work.

The Broker is only trusted with verifying that the Publisher is authorized to publish, but is not trusted with the publications itself, which it cannot read nor modify. In this setting, caching of publications on the Broker is still allowed.

TODO: expand on security and Privacy considerations

7. IANA Considerations

TODO: "key" parameter, OSCON profile identifier

8. Acknowledgments

The author wishes to thank John Mattsson, Ludwig Seitz and Goeran Selander for the useful discussion that helped shape this document.

9. References

9.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-05 (work in progress), February 2017.

[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-00 (work in progress), October 2016.

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

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

9.2. Informative References

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

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-09 (work in progress), October 2016.

[I-D.seitz-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-seitz-ace-oauth-authz-00 (work in progress), October 2015.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

Author's Address

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 27, 2017

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

April 25, 2017

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-06

Abstract

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a compact, and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys that can be used over any layer. EDHOC messages are encoded with CBOR and COSE, allowing reuse of existing libraries.

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 October 27, 2017.

Table of Contents

1. Introduction

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs] or where the protocol needs to work on a variety of underlying protocols. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and energy [RFC7228]. A method for protecting individual messages at the application layer suitable for constrained devices, is provided by CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg]), which builds on the Concise Binary Object Representation (CBOR) [RFC7049].

In order for a communication session to provide forward secrecy, the communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol with ephemeral keys, from which shared key material can be derived. This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), an authenticated ECDH protocol using CBOR and COSE objects. Authentication is based on credentials established out of band, e.g. from a trusted third party, such as an Authorization Server as specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports authentication using pre-shared keys (PSK), raw public keys (RPK), and certificates (Cert). Note that this document focuses on authentication and key establishment: for integration with authorization of resource access, refer to [I-D.seitz-ace-oscoap-profile]. This document also specifies the derivation of shared key material.

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56a [SP-800-56a], and HKDF [RFC5869]. CBOR [RFC7049] and COSE [I-D.ietf-cose-msg] are used to implement these standards.

1.1. Terminology

This document use the same informational CBOR Data Definition Language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl] grammar as COSE (see Section 1.3 of [I-D.ietf-cose-msg]). A vertical bar | denotes byte string concatenation.

1.2. Requirements Language

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

2. Protocol Overview

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and TLS 1.3, EDHOC is built on a variant of the SIGMA protocol which provide identity protection, and like TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an AEAD algorithm is shown in Figure 1.

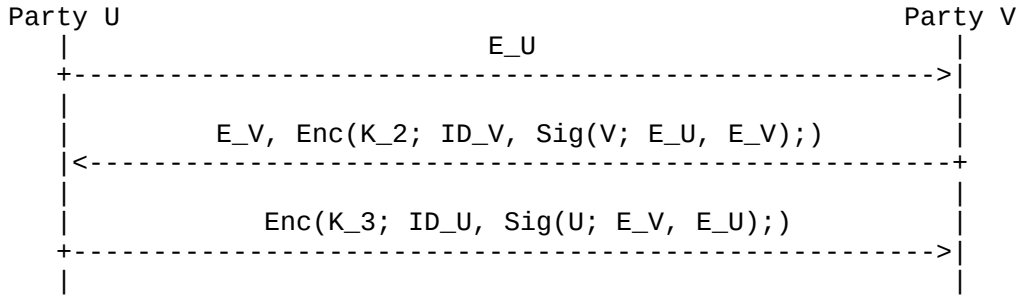


Figure 1: AEAD variant of the SIGMA-I protocol

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive the keying material. The messages are signed, MACed, and encrypted.

- E_U and E_V are the ECDH ephemeral public keys of U and V, respectively.
- ID_U and ID_V are identifiers for the public keys of U and V, respectively.
- Sig(U; .) and S(V; .) denote signatures made with the private key of U and V, respectively.
- Enc(K; P; A) denotes AEAD encryption of plaintext P and additional authenticated data A using the key K derived from the shared secret. The AEAD MUST NOT be replaced by plain encryption, see Section 8.

As described in Appendix B of [SIGMA], in order to create a "full-fledge" protocol some additional protocol elements are needed. EDHOC adds:

- Explicit session identifiers S_U, S_V chosen by U and V, respectively.
- Explicit nonces N_U, N_V chosen freshly and anew with each session by U and V, respectively.
- Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.

EDHOC also makes the following additions:

- Negotiation of key derivation, encryption, and signature algorithms:
  - U proposes one or more algorithms of the following kinds:
    - HKDF
    - AEAD
    - Signature verification
    - Signature generation
  - V selects one algorithm of each kind
- Verification of common preferred ECDH curve:
  - U lists supported ECDH curves in order of preference
  - V verifies that the ECDH curve of the ephemeral key is the most preferred common curve
- Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR and COSE libraries. EDHOC does not put any requirement on the lower layers and can therefore be also be used e.g. in environments without IP.

This paper is organized as follows: Section 3 specifies general properties of EDHOC, including formatting of the ephemeral public keys and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, and Appendix A provides a wealth of test vectors to ease implementation and ensure interoperability.

3. EDHOC Overview

EDHOC consists of three messages (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consists of a CBOR array where the first element is an int specifying the message type (MSG_TYPE). After creating EDHOC message_3, Party U can derive the traffic key (master secret) and protected application data can therefore be sent

in parallel with EDHOC message_3. The application data may e.g. be protected using the negotiated AEAD algorithm. EDHOC may be used with the media type application/edhoc defined in Section 7.

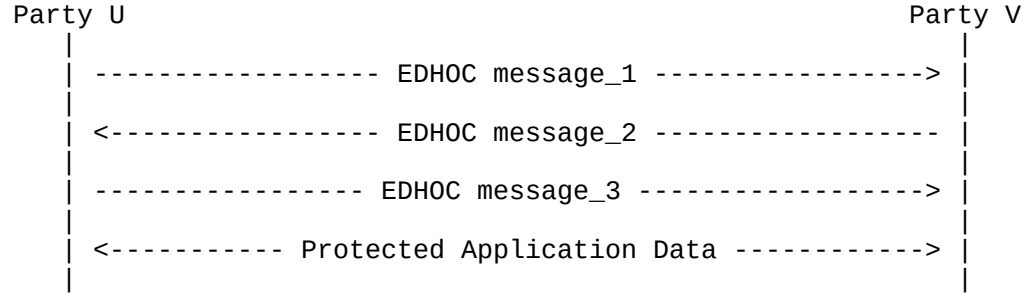


Figure 2: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or certificates (Cert). EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed.

EDHOC also allows application data (APP_1, APP_2, APP_3) to be sent in the respective messages. APP_1 is unprotected, APP_2 is protected (encrypted and integrity protected), and APP_3 is protected and mutually authenticated. When EDHOC is used with asymmetric key authentication APP_2 is sent to an unauthenticated party, but with symmetric key authentication APP_2 is mutually authenticated.

3.1. Formatting of the Ephemeral Public Keys

The ECDH ephemeral public key SHALL be formatted as a COSE_Key of type EC2 or OKP according to section 13.1 and 13.2 of [I-D.ietf-cose-msg]. The curve X25519 is mandatory to implement. For Elliptic Curve Keys of type EC2, point compression is mandatory to implement.

3.2. Key Derivation

Key and IV derivation SHALL be done as specified in Section 11.1 of [I-D.ietf-cose-msg] with the following input:

- The PRF SHALL be the HKDF [RFC5869] in the ECDH-SS w/ HKDF negotiated during the message exchange (HKDF_V).

- The secret SHALL be the ECDH shared secret as defined in Section 12.4.1 of [I-D.ietf-cose-msg].
- salt = PSK / nil
- The context information SHALL be the serialized COSE_KDF_Context with the following values:
  - AlgorithmID = tstr / int
  - PartyInfo = (nil, nil, nil)
  - SuppPubInfo SHALL contain:
    - keyDataLength int
    - protected SHALL be a zero length bstr
    - other = aad_2 / aad_3 / exchange_hash

exchange_hash = bstr

where exchange_hash, in diagnostic non-normative notation, is:

exchange_hash = H(H(message_1 | message_2) | message_3)

where H() is the hash function in HKDF_V, and | denotes byte string concatenation.

The salt SHALL only be present in the symmetric case.

Symmetric keys and IVs SHALL be derived with the negotiated PRF (HKDF_V) and with the secret set to the ECDH shared secret.

For message_i the key and IV, called K_i and IV_i, SHALL be derived using other = aad_i, where i = 2 or 3. The key SHALL be derived using AlgorithmID set to the negotiated AEAD (AEAD_V), and keyDataLength equal to the key length of AEAD_V. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in section 12.1.2. of [I-D.ietf-cose-msg], and keyDataLength equal to the IV length of AEAD_V.

Application specific traffic keys and other data SHALL be derived using other = exchange_hash. AlgorithmID is defined by the application and SHALL be different for different data being derived (an example is given in Appendix C.2). keyDataLength is set to the length of the data being derived.

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

EDHOC supports authentication with raw public keys (RPK) and certificates (Cert) with the requirements that:

- Party U's SHALL be able to identify Party V's public key using ID_V.
- Party V's SHALL be able to identify Party U's public key using ID_U.

ID_U and ID_V either enable the other party to retrieve the public key (kid, x5t, x5u) or they contain the public key (x5c), see [I-D.schaad-cose-x509]. Party U and party V MAY use different type of credentials, e.g. one uses RPK and the other Cert. Party U and party V MAY use different signature algorithms.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

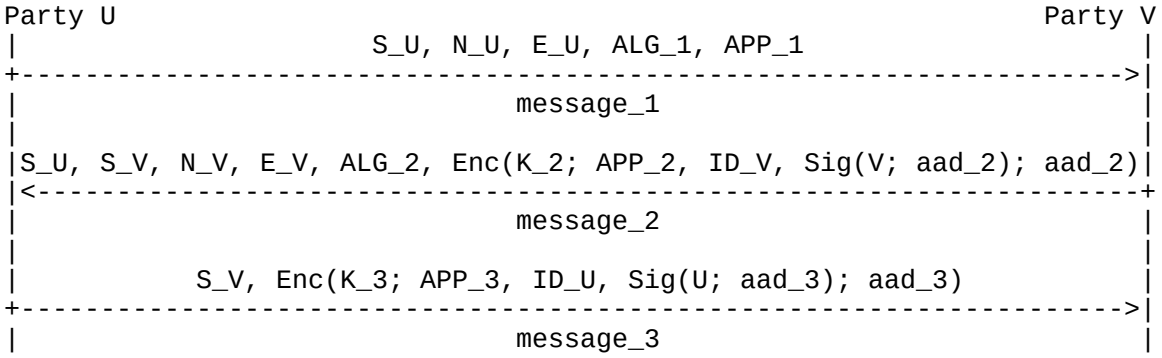


Figure 3: EDHOC with asymmetric key authentication.

4.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with asymmetric keys, the COSE algorithms ECDH-SS + HKDF-256, AES-CCM-64-64-128, and EdDSA are mandatory to implement.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  SIGs_V : alg_array,
  SIGs_U : alg_array,
  ? APP_1 : bstr
]

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 1
- S_U - variable length session identifier
- N_U - 64-bit random nonce
- E_U - the ephemeral public key of Party U
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- HKDFs_U - supported ECDH-SS w/ HKDF algorithms
- AEADs_U - supported AEAD algorithms
- SIGs_V - signature algorithms, with which Party U supports verification
- SIGs_U - signature algorithms, with which Party U supports signing
- APP_1 - bstr containing application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_U

- Choose a session identifier S_U and store it for the length of the protocol.

- Format message_1 as specified in Section 4.2.1.

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.

- Verify that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message MUST include the following diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U:

ERR_MSG = "Curve not supported; X"

where X is the first curve in ECDH-Curves_U that V supports, encoded as in Table 22 of {{I-D.ietf-cose-msg}}.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr,
  SIG_V : int / tstr,
  SIG_U : int / tstr
)

aad_2 = bstr
```

where aad_2, in diagnostic non-normative notation, is:

```
aad_2 = message_1 | [ data_2 ] | ? Cert_V
```

where:

- MSG_TYPE = 2
- S_V - variable length session identifier
- N_V - 64-bit random nonce
- E_V - the ephemeral public key of Party V
- HKDF_V - a single chosen algorithm from HKDFs_U
- AEAD_V - a single chosen algorithm from AEADs_U
- SIG_V - a single chosen algorithm from SIGs_V with which Party V signs
- SIG_U - a single chosen algorithm from SIGs_U with which Party U signs

- COSE_ENC_2 has the following fields and values:
  - external_aad = aad_2
  - plaintext = [COSE_SIG_V, ? APP_2]
- COSE_SIG_V is a COSE_Sign1 object with the following fields and values:
  - unprotected = { xyz: ID_V }
  - detached payload = aad_2
- xyz - any COSE map label that can identify a public key, see Section 4.1
- ID_V - identifier for the public key of Party V
- APP_2 - bstr containing application data
- Cert_V - The end-entity certificate of Party V
- H() - the hash function in HKDF_V

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.
- Generate the pseudo-random nonce N_V
- Choose a session identifier S_V and store it for the length of the protocol.
- Select HKDF_V, AEAD_V, SIG_V, and SIG_U from the algorithms proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.
- Format message_2 as specified in Section 4.3.1:
  - COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the private key of Party V.

* COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

  + If certificates are used then aad_2 MUST include Cert_V

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.
- Verify that HKDF_V, AEAD_V, SIG_V, and SIG_U were proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.
- Verify (OPTIONAL) that N_V has not been received before.
- Verify message_2 as specified in Section 4.3.1:
  * COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.
  * COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the public key of Party V.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 = bstr
```

where aad_3, in diagnostic non-normative notation, is:

aad_3 = H(message_1 | message_2) | [data_3] | ? Cert_U

where:

- MSG_TYPE = 3
- COSE_ENC_3 has the following fields and values:
  - external_aad = aad_3
  - plaintext = [COSE_SIG_U, ? APP_3]
- COSE_SIG_U is a COSE_Sign1 object with the following fields and values:
  - unprotected = { xyz: ID_U }
  - detached payload = aad_3
- xyz - any COSE map label that can identify a public key, see Section 4.1
- ID_U - identifier for the public key of Party U
- APP_3 - bstr containing application data
- Cert_U - The end-entity certificate of Party U

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 4.4.1:
  - COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the private key of Party U.
  - COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.
    + If certificates are used then aad_3 MUST include Cert_U

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.
- Verify message_3 as specified in Section 4.4.1.
  - COSE_Encrypt0 is decrypted as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.
  - COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the public key of Party U;

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

EDHOC supports authentication with pre-shared keys. Party U and V are assumed to have a pre-shared uniformly random key (PSK) with the requirement that:

- Party V SHALL be able to identify the PSK using KID.

KID either enable the other party to retrieve the PSK or contain the PSK (e.g. CBOR Web Token).

EDHOC with symmetric key authentication is illustrated in Figure 4.

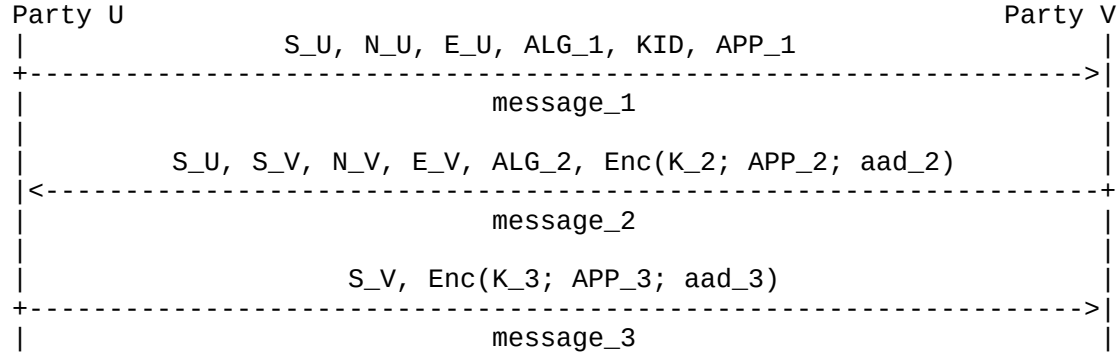


Figure 4: EDHOC with symmetric key authentication.

5.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with symmetric keys, the COSE algorithms ECDH-SS + HKDF-256 and AES-CCM-64-64-128 are mandatory to implement.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  data_1
]

data_1 = (
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  KID : bstr,
  ? APP_1 : bstr
)

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 4
- S_U - variable length session identifier
- N_U - 64-bit random nonce
- E_U - the ephemeral public key of Party U
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- HKDFs_U - supported ECDH-SS w/ HKDF algorithms
- AEADs_U - supported AEAD algorithms

- KID - identifier of the pre-shared key
- APP_1 - bstr containing application data

5.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.
- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.
- Generate the pseudo-random nonce N_U
- Choose a session identifier S_U and store it for the length of the protocol.
- Format message_1 as specified in Section 5.2.1.

5.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.
- Verify that at least one of each kind of the proposed algorithms are supported.
- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message SHOULD include a diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U.

5.3. EDHOC Message 2

5.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr
)
```

aad_2, in diagnostic non-normative notation, is:

```
aad_2 = message_1 | [ data_2 ]
```

where:

- MSG_TYPE = 5
- S_V - variable length session identifier
- N_V - 64-bit random nonce
- E_V - the ephemeral public key of Party V
- HKDF_V - an single chosen algorithm from HKDFs_U
- AEAD_V - an single chosen algorithm from AEADs_U
- COSE_ENC_2 has the following fields and values:
  - external_aad = aad_2
  - plaintext = ? APP_2
- APP_2 - bstr containing application data
- H() - the hash function in HKDF_V

5.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.
- Generate the pseudo-random nonce N_V
- Choose a session identifier S_V and store it for the length of the protocol.
- Select HKDF_V and AEAD_V from the algorithms proposed in HKDFs_U and AEADs_U.
- Format message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

5.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.
- Verify message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)
```

aad_3, in diagnostic non-normative notation, is:

```
aad_3 = H( message_1 | message_2 ) | [ data_3 ]
```

where:

- MSG_TYPE = 6
- COSE_ENC_3 has the following fields and values:
  - external_aad = aad_3
  - plaintext = ? APP_3
- APP_3 - bstr containing application data

5.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

5.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.
- Verify message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is decrypted and verified as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

6. Error Handling

6.1. Error Message Format

This section defines a message format for an EDHOC error message, used during the protocol. This is an error on EDHOC level and is independent of the transport layer used. An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR array as defined below

```
error = [
  MSG_TYPE : int,
  ? ERR_MSG : tstr
]
```

where:

- MSG_TYPE = 0
- ERR_MSG is an optional text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252].

7. IANA Considerations

7.1. Media Types Registry

IANA has added the media type 'application/edhoc' to the Media Types registry:

Type name: application

Subtype name: edhoc

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of this document.

Interoperability considerations: N/A

Published specification: [[this document]] (this document)

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

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

Person & email address to contact for further information:
Goeran Selander <goran.selander@ericsson.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander <goran.selander@ericsson.com>

Change Controller: IESG

8. Security Considerations

EDHOC builds on the SIGMA-I family of theoretical protocols that provides perfect forward secrecy and identity protection with a minimal number of messages. The encryption algorithm of the SIGMA-I protocol provides identity protection, but the security of the protocol requires the MAC to cover the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be

replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

EDHOC adds an explicit message type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. EDHOC uses the same Sign-then-MAC approach as TLS 1.3.

EDHOC does not include negotiation of parameters related to the ephemeral key, but it enables Party V to verify that the ECDH curve used in the protocol is the most preferred curve by U which is supported by both U and V.

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to APP_1 and APP_2 in the asymmetric case, and APP_1 and KID in the symmetric case. The communicating parties may therefore anonymize KID.

Using the same KID or unprotected application data in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported algorithms may be used to identify the application.

Party U and V must make sure that unprotected data does not trigger any harmful actions. In particular, this applies to APP_1 in the asymmetric case, and APP_1 and KID in the symmetric case. Party V should be aware that replays of EDHOC message_1 cannot be detected unless previous nonces are stored.

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. If ECDSA is supported, "deterministic ECDSA" as specified in RFC6979 is RECOMMENDED.

Nonces MUST NOT be reused, both parties MUST generate fresh random nonces.

Ephemeral keys SHOULD NOT be reused, both parties SHOULD generate fresh random ephemeral key pairs. Party V MAY reuse the ephemeral key to limit the effect of certain DoS attacks. For example, to reduce processing costs in the case of repeated uncompleted protocol runs, party V MAY pre-compute its ephemeral key E_V and reuse it for a small number of concurrent EDHOC executions, for example until a number of EDHOC protocol instances has been successfully completed,

which triggers party V to pre-compute a new ephemeral key E_V to use with subsequent protocol runs.

The referenced processing instructions in [SP-800-56a] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported.

The choice of key length used in the different algorithms needs to be harmonized, so that a sufficient security level is maintained for certificates, EDHOC, and the protection of application data. Party U and V should enforce a minimum security level.

Note that, depending on the application, the keys established through the EDHOC protocol will need to be renewed, in which case the communicating parties need to run the protocol again.

Implementations should provide countermeasures to side-channel attacks such as timing attacks.

9. Acknowledgments

The authors want to thank Jim Schaad for reviewing intermediate versions and for contributing many concrete proposals incorporated in this version. We are also greatful to Ilari Liusvaara and Ludwig Seitz for reviewing previous versions of the draft.

TODO: This section should be after Appendixes and before Author’s address according to RFC7322.

10. References

10.1. Normative References

[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.schaad-cose-x509]
Schaad, J., "CBOR Encoded Message Syntax (COSE): Headers for carrying and referencing X.509 certificates", draft-schaad-cose-x509-00 (work in progress), November 2016.

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

[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>.

[SIGMA] Krawczyk, H., "SIGMA - The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols (Long version)", June 2003, <http://webee.technion.ac.il/~hugo/sigma-pdf.pdf>.

[SP-800-56a] Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2, May 2013, <http://dx.doi.org/10.6028/NIST.SP.800-56Ar2>.

10.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.hartke-core-e2e-security-reqs] Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-02 (work in progress), January 2017.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

[I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-02 (work in progress), March 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

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

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

[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>.

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. PSK Chaining

An application using EDHOC with symmetric keys may have a security policy to change the PSK as a result of successfully completing the EDHOC protocol. In this case, the old PSK SHALL be replaced with a new PSK derived using other = exchange_hash, AlgorithmID = "EDHOC PSK Chaining" and keyDataLength equal to the key length of AEAD_V, see Section 3.2.

Appendix C. EDHOC with CoAP and OSCOAP

C.1. Transferring EDHOC in CoAP

EDHOC can be transferred as an exchange of CoAP [RFC7252] messages, with the CoAP client as party U and the CoAP server as party V. By default EDHOC is sent to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discorvered e.g. using resource directory [I-D.ietf-core-resource-directory].

In practice, EDHOC message_1 is sent in the payload of a POST request from the client to the server's resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response. EDHOC message_3 or the EDHOC error message is sent from the client to the server's resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response

An example of successful EDHOC exchange using CoAP is shown in Figure 5.

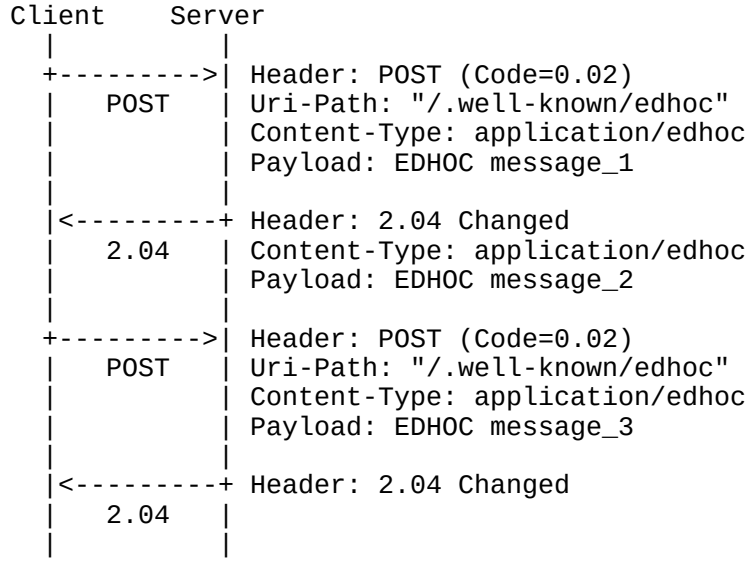


Figure 5: Transferring EDHOC in CoAP

C.2. Deriving an OSCOAP context from EDHOC

When EDHOC is use to derive parameters for OSCOAP [I-D.ietf-core-object-security], the parties must make sure that the EDHOC session identifiers are unique Recipient IDs in OSCOAP. In case that the CoAP client is party U and the CoAP server is party V:

- The AEAD Algorithm is AEAD_V, as defined in this document
- The KDF algorithm is HKDF_V, as defined in this document
- The Client's Sender ID is S_V, as defined in this document
- The Server's Sender ID is S_U, as defined in this document

- The Master Secret is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Secret" and keyDataLength equal to the key length of AEAD_V.

- The Master Salt is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Salt" and keyDataLength equal to 64 bits.

Authors' Addresses

Goeran Selander
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

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

G. Selander
Ericsson AB
S. Raza
RISE SICS
M. Vucinic
Inria
M. Furuhed
Nexus
M. Richardson
Sandelman Software Works
March 13, 2017

Enrollment with Application Layer Security
draft-selander-ace-eals-00

Abstract

This document specifies public key certificate enrollment procedures authenticated with application-layer security protocols suitable for Internet of Things deployments. The protocols leverage existing IoT standards including Constrained Application Protocol (CoAP), Concise Binary Object Representation (CBOR) and the CBOR Object Signing and Encryption (COSE) format.

Table of Contents

1. Introduction

Asymmetric cryptography with Public Key Infrastructure (PKI) is a de-facto key exchange and mutual authentication solution in the Internet. Though solutions based on PSK are still state-of-the-art in sensor networks they are not scalable to Internet-connected billions of things. Therefore, most IoT security standards support asymmetric cryptographic protocols. The greatest challenge with asymmetric cryptography and PKI is enrollment, the process of certifying keys. Enrollment is even more challenging in the IoT as things are resource-constrained and traditional enrollment techniques are not compatible with recent IoT security protocols. Without secure enrollment, PKI will not be trustworthy and in turn the

cybersecurity of the entire system will be at stake even though the underlying cryptographic cipher suites are most secure.

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, in particular in constrained environments [RFC7228] and when using CoAP [RFC7252]; for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs], or where it is beneficial that the security protocol is independent of lower layers, such as when securing CoAP over mixed transport protocols.

Application layer security protocols suitable for constrained devices are in development, including the secure communication protocol OSCOAP [I-D.ietf-core-object-security]. OSCOAP defines an extension to the Constrained Application Protocol (CoAP) providing encryption, integrity and replay protection end-to-end between CoAP client and server based on a shared secret. The shared secret can be established in different ways e.g. using a trusted third party such as in ACE [I-D.ietf-ace-oauth-authz], or using a key exchange protocol such as EDHOC [I-D.selander-ace-cose-ecdhe]. OSCOAP and EDHOC can leverage other constrained device primitives developed in the IETF: CoAP, CBOR [RFC7049] and COSE [I-D.ietf-cose-msg], and makes only a small additional implementation footprint.

Lately, there has been a discussion in several IETF working groups about certificate enrollment protocols suitable for IoT devices, to support the use case of an IoT device joining a new network domain and establishing credentials valid in this domain. This document describes Enrollment with Application Layer Security (EALS), a certificate enrollment protocol based on CMC [RFC5272] and using OSCOAP as a secure channel. This document also describes how ACE and EDHOC can be used for establishing an authenticated and authorized channel.

This work is inspired by the Enrollment over Secure Transport (EST) protocol [RFC7030], which also is based on CMC, but EALS is secured on application layer instead of on transport layer.

1.1. Terminology

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

2. CMC protocol

2.1. Simple Enrollment

This section describes the simple enrollment protocol, which is an embedding of the Simple PKI Request/Response protocol of CMC [RFC5272] in Object Secure CoAP (OSCOAP) [I-D.ietf-core-object-security].

The simple enrollment protocol is a 2-pass protocol between an EALS client (e.g. an IoT device) and an EALS server (a Certification Authority (CA)), see Figure 1. The protocol assumes that both EALS client and EALS server implement CoAP and the Object-Security option of CoAP (OSCOAP).

OSCOAP assumes the existence of a shared secret between an EALS client and server. The shared secret may be obtained by running a key agreement algorithm or by an aid of a trusted third party. Mutual authentication and authorization occurs during this key agreement stage. The simple enrollment protocol may also be run directly with a pre-shared key. In that case, authentication and authorization of EALS client and server is implicit to the shared key protecting the /eals resource.

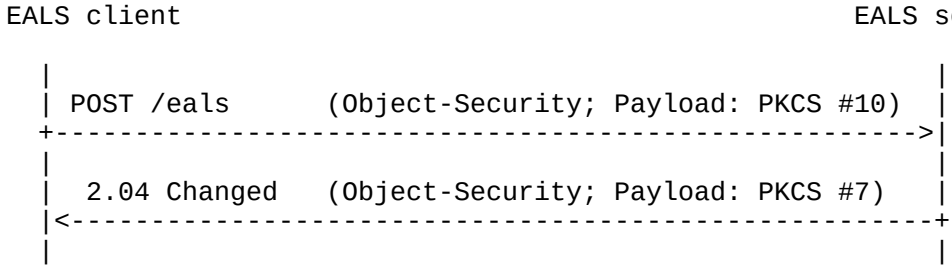


Figure 1: The Simple Enrollment Protocol.

The simple enrollment protocol consists of a CoAP message exchange.

The EALS client sends a CoAP request:

- Method is POST
- Uri-Path is "eals"
- Object-Security option is present

- Payload is the CMC Simple PKI Request [RFC5272] (i.e. a PKCS #10 certification request).

If successful, the EALS server sends a CoAP response:

- Code is 2.04 (Changed)
- Content-Format is "application/pkcs7-mime" (TBD)
- Object-Security option is present
- Payload is a certs-only CMC Simple PKI Response [RFC5272] (i.e the issued certificate)

OSCOAP protects the CoAP message exchange between the endpoints over any transport and via intermediary nodes. The OSCOAP protection requires that a security context is established between client and server. The security context can be derived from a set of Input Parameters (Section 3.3 of [I-D.ietf-core-object-security]), including at least the following:

- Master Secret
- Sender ID
- Recipient ID

where the Master Secret is a uniformly random byte string, and the Sender ID and Recipient ID are byte strings identifying the endpoints. In Section 3 we give examples of how to the OSCOAP input parameters can be established.

The server MUST verify that the Master Secret is associated to the Distinguished Name for which the client is requesting a certificate.

Note 1: The encodings and formats used by CMC may later be updated with other equivalents more adapted to constrained environments.

2.2. Re-enrollment

Re-enrollment and re-keying of clients occurs using the same exchange as during the simple enrollment protocol. Re-enrollment request follows the same format as during the simple enrollment. In case of success, re-enrollment response contains certs-only CMC Simple PKI Response, as in the case of simple enrollment with content-format set to "application/pkcs7-mime".

TBD. Requirements on parsing PKCS messages and X.509 certificates

TBD. Error handling with CoAP error codes

TBD. Server-side key generation

2.3. Full Enrollment

It is straightforward to extend the simple enrollment to the CMC Full PKI Request/Response protocol.

In this case, to authorize the PKCS#10 request to the CA, it is enveloped in a CMC message and signed with a pre-installed device private key and certificate by the device itself.

The public key in the device certificate acts as a unique identifier of the device. By trusting the CA issuing the pre-installed certificate, the enrolment CA can acknowledge the signed request. The trusted factory CA will also ensure the origin of the device.

An alternative to authorize the PKCS#10 request to the CA, is to use a security gateway that can envelope the request in a CMC message using a certificate trusted by the CA.

The details are FFS.

2.4. Compiling Certificate Content

A CA have several means of compiling certificate content during issuance. The subject Distinguished Name (DN) information for the certificate may be based on the content of the request alone.

Alternatively, complementary data can be added to the request by the CA from an external source trusted by the CA, or taken from records of pre-registered information on end-entities that is stored in the CA system and which can be uniquely matched to the data in the request. Due to the constrained device capabilities the amount of subject DN data in a request may be very limited. The method of adding complementary data for the certificate can be a choice of the CA, assuming the source of complementary data can be provided in a trustworthy way.

With the option to add complementary data to a certificate request, the end-entity provided data can be diminished by e.g. submitting only the public key in the PKCS#10 content. The public key can be used to match the device to pre-registered data or for retrieval of subject data from other sources.

3. Establishment of OSCOAP Input Parameters

In this section we present two application layer protocols for establishing OSCOAP input parameters (Section 3.3 of [I-D.ietf-core-object-security]), in particular the OSCOAP master secret.

3.1. EDHOC

EDHOC [I-D.selander-ace-cose-ecdhe] is a key establishment protocol, corresponding to the handshake protocol of TLS, encoded with CBOR and using COSE that may be transported with e.g. CoAP. EDHOC provides mutual authentication of client and server and establishes a shared secret with forward secrecy which may be used as OSCOAP master secret in the CMC protocol (Section 2).

The asymmetric keys authenticated version of EDHOC is described in section 4 of [I-D.selander-ace-cose-ecdhe], a simplified version of the protocol is shown in Figure 2.

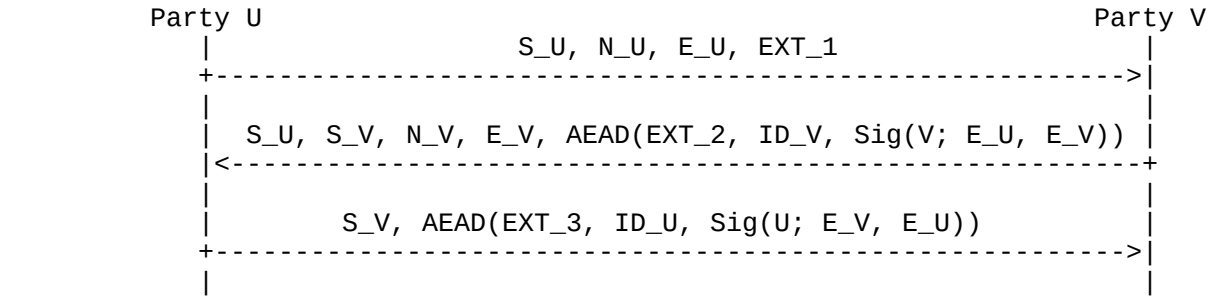


Figure 2: EDHOC with asymmetric key authentication (simplified). S = session identifer, N = nonce, E = ephemeral public key, ID = identifier, and EXT = application defined extension.

The session identifiers S_U and S_V may be used as OSCOAP input parameters Sender ID and Recipient ID of party U, and v.v. as described in Appendix B2 of [I-D.selander-ace-cose-ecdhe].

Figure 3 shows an example of using the EDHOC protocol to establish a mutually authenticated and authorized channel for the simple enrolment protocol. In this case the EALS server is EDHOC client (the mapping with interchanged roles is straightforward and left FFS). This setting has the following properties:

1. The EALS server initiates the EDHOC protocol. This allows the EALS server to orchestrate many concurrent enrollments, and control the associated network load.

2. The EALS client is authenticated first (EDHOC message_2). This allows the EALS server to authenticate the EALS client, and with this information to authorize the EALS client before completing the EDHOC protocol. The EALS server may in this case also relay authorizaton information about the EALS client, such as an ownership voucher, to the client in EDHOC extension EXT_3.

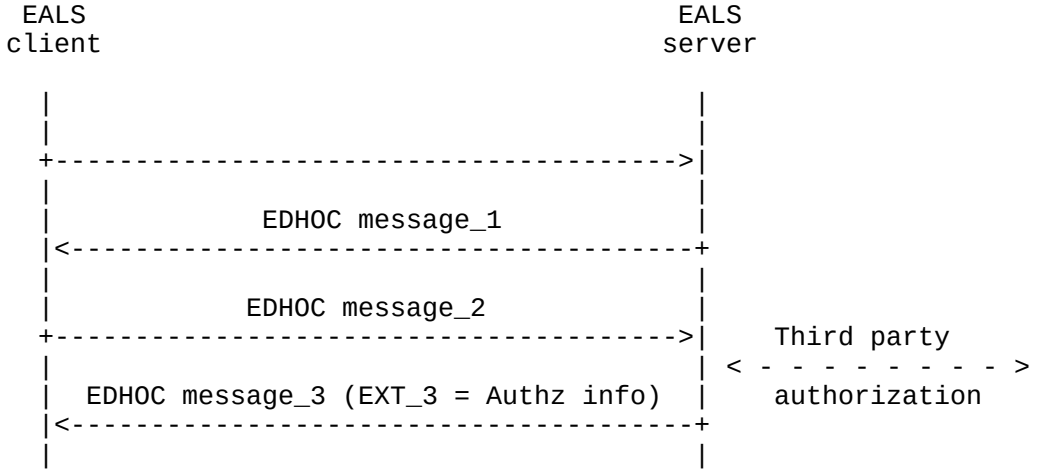


Figure 3: EALS extension of EDHOC.

Appendix B1 of [I-D.selander-ace-cose-ecdhe] shows how to embed EDHOC in a CoAP message exchange, a similar embedding can be applied here.

TBD Detail the protocol

3.2. ACE

The ACE protocol framework [I-D.ietf-ace-oauth-authz] is an adaptation of OAuth 2.0 to IoT deployments. ACE describes different message flows for a Client to get authorized access to a Resource Server (RS) by leveraging an Authorization Server (AS).

The Token Introspection flow (Section 7 of [I-D.ietf-ace-oauth-authz]) allows an RS to access authorization information relating to a client provided Access Token. If the access token is valid, the RS obtains information about the access rights and a symmetric key used by the client, and also a Client

Token containing the same shared key protected for the legitimate client (Section 7.4 of [I-D.ietf-ace-oauth-authz], Figure 4).

This message flow assumes that the Client and AS, as well as the RS and AS, has or can establish a mutually authenticated secure channel such that:

- The AS can encrypt the symmetric key for the Client in the Client Token, and the Client can verify the Client Token is issued by the AS;
- The RS and AS can exchange encrypted, integrity and replay protected introspection messages. In this case, the establishment of the secure channel can take place immediately before introspection, triggered by the RS receiveing the Access Token.

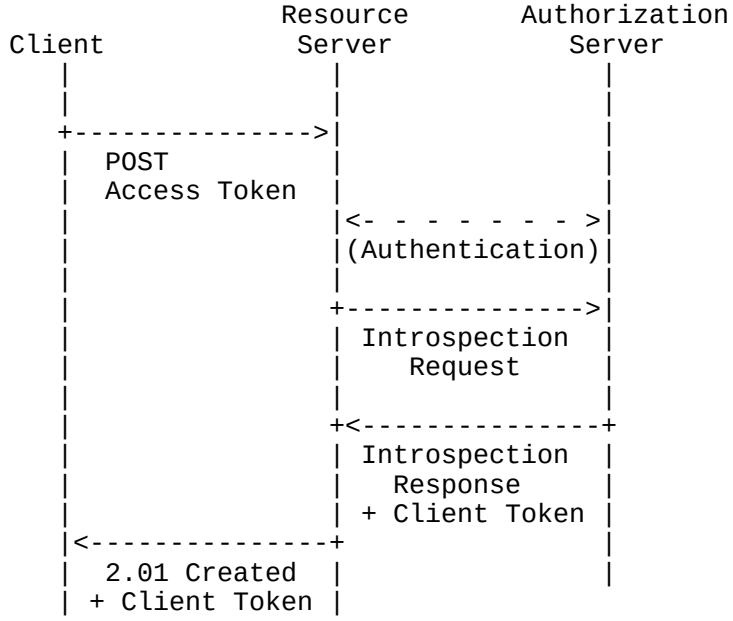


Figure 4: ACE Token Introspection with Client Token.

By mapping the EALS client and server to the ACE client and resource server, respectively, this application of ACE enables the authorization of EALS client and establishment of a shared key, which can be used as master secret with OSCOAP in the CMC protocol (Section 2). In this case, the access token contains access rights to /eals, but is not (yet) bound to a particular resource server.

The access token could be pre-provisioned to the client, e.g. during manufacture. Information about binding to resource server comes with the introspection response.

Section 2 of [I-D.seitz-ace-oscoap-profile] defines additional common header parameters for COSE_Key structure that are used to carry OSCOAP input parameters Sender and Recipient ID. The OSCOAP master secret is transported as part of the symmetric COSE_Key object. This document uses the same construct: COSE_Key object with OSCOAP input parameters present is transported as part of the Introspection Response and in the Client Token.

For the benefit of the client authorizing the enrollment, this document defines an additional common parameter for the Client Token called Voucher, extending the definition in Section 7.4 of [I-D.ietf-ace-oauth-authz]:

voucher
: OPTIONAL. Contains authorization information about the server, e.g. ownership voucher. The encoding is TBD.

Parameter name	CBOR Key	Major Type
voucher	TBD	2 (byte string)

Figure 5: CBOR mapping of parameters extending the client token.

Additionally, the certificate attributes presented by the Client in the enrolment request (Section 2) may be carried in the Client Token. The encoding is TBD.

4. Application to 6TiSCH

One candidate embedding of EALS into a bootstrapping architecture is as described in [I-D.ietf-6tisch-minimal-security]. The new device (a.k.a. Pledge) requests to be admitted into the network managed by the Join Registrar/Coordinator. The Pledge maps to an EALS/CoAP client, and the Join Registrar/Coordinator maps to an EALS/CoAP server.

When a pledge first joins a constrained network, it typically does not have IPv6 connectivity to reach the Join Registrar/Coordinator. For that reason, pledge communicates with the Join Proxy, a one hop neighbor of the pledge. Join Proxy statelessly relays the exchanges between the pledge and the Join Registrar/Coordinator.

As in the model of [I-D.ietf-6tisch-minimal-security], the Join Proxy plays the role of a CoAP proxy. Default CoAP proxy, however, keeps state information in order to relay the response back to the originating client, in this case the pledge. To mitigate Denial of Service attacks at the Join Proxy, [I-D.ietf-6tisch-minimal-security] mandates the use of a new CoAP option, Stateless-Proxy option, that allows the Join Proxy to operate without keeping per-client state.

The use of EDHOC as described in Section 3.1 enables mutual authentication and authorization of Pledge and Join Registrar/ Coordinator, and supports the use of the Stateless-Proxy option in order to provide the CoAP Proxy functionality described in this section.

5. Application to BRSKI

Another application of EALS is to the BRSKI [I-D.ietf-anima-bootstrapping-keyinfra] problem statement. BRSKI specifies an automated bootstrapping of a remote secure key infrastructure (BRSKI) using vendor installed X.509 certificate, in combination with a vendor authorized service on the Internet. BRSKI is referencing Enrolment over Secure Transport (EST) [RFC7030] to enable zero-touch joining of a device in a network domain. The same approach can be applied using EDHOC instead of EST, as is outlined in this document.

The audit/ownership vouchers specified in [I-D.ietf-anima-bootstrapping-keyinfra] are carried as part of EDHOC application-defined extensions, as described in Section 3.1. Nonces of the EDHOC protocol can be used for freshness also of the authorization step.

The limitations of applicability to energy-constrained devices due to credential size applies also to this document, and further work is needed to specify certificate formats relevant to constrained devices. Having said that, one rationale for this document is a more optimized message exchange, and potentially also code footprint, which is favorable in low-power deployments.

6. Security Considerations

7. Privacy Considerations

8. IANA Considerations

9. Acknowledgments

The authors wants to thank the participants of the 6tisch security design team for discussions and input contributing to this document.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-05 (work in progress), February 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[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.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-04 (work in progress), October 2016.

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

[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>.

10.2. Informative References

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-02 (work in progress), January 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., and K. Pister, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-01 (work in progress), February 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-04 (work in progress), October 2016.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <http://www.rfc-editor.org/info/rfc5272>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <http://www.rfc-editor.org/info/rfc7030>.

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

Appendix A. Examples

Authors' Addresses

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

Email: goran.selander@ericsson.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: shahid.raza@ri.se

Malisa Vucinic
Inria
2 Rue Simone Iff
Paris 75012
France

Email: malisa.vucinic@inria.fr

Martin Furuhed
Nexus
Telefonv. 26
Stockholm SE-12626
Sweden

Email: martin.furuhed@nexusgroup.com

Michael Richardson
Sandelman Software Works
470 Dawson Avenue
Ottawa, ON K1Z5V7
Canada

Email: mcr+ietf@sandelman.ca

ACE
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2017

S. Kumar
Philips Lighting Research
P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
M. Furuhed
Nexus Group
S. Raza
RISE SICS
March 9, 2017

EST over secure CoAP (EST-coaps)
draft-vanderstok-ace-coap-est-01

Abstract

Low-resource devices in a Low-power and Lossy Network (LLN) can operate in a mesh network using the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) and IEEE 802.15.4 link-layer standards. Provisioning these devices in a secure manner with keys (often called secure bootstrapping) used to encrypt and authenticate messages is the subject of Bootstrapping of Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra] and 6tisch Secure Join [I-D.ietf-6tisch-dtsecurity-secure-join]. Enrollment over Secure Transport (EST) [RFC7030], based on TLS and HTTP, is used in BRSKI. Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) [RFC7252] for message exchanges. This document defines how low-resource devices are expected to use EST over secure CoAP (EST-coaps) for secure bootstrapping and certificate enrollment. 6LoWPAN fragmentation management and minor extensions to CoAP are needed to enable EST-coaps.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2017.

Table of Contents

1. Introduction

IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [RFC4944] on IEEE 802.15.4 [ieee802.15.4] wireless networks is becoming common in many industry application domains such as lighting controls. However, commissioning of such networks suffers from a lack of standardized secure bootstrapping mechanisms for these networks.

Although IEEE 802.15.4 defines how security can be enabled between nodes within a single mesh network, it does not specify the provisioning and management of the keys. Therefore, securing a 6LoWPAN network with devices from multiple manufacturers with different provisioning techniques is often tedious and time consuming.

Bootstrapping of Remote Secure Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra] addresses the issue of bootstrapping networked devices in the context of Autonomic Networking Integrated Model and Approach (ANIMA). [I-D.ietf-6tisch-minimal-security] and [I-D.ietf-6tisch-dtsecurity-secure-join] also address secure bootstrapping in the 6tisch context targeted to low-resource devices. BRSKI has not been developed specifically for low-resource devices in constrained networks. Constrained networks use DTLS [RFC6347], CoAP [RFC7252], and UDP instead of TLS [RFC5246], HTTP [RFC7230] and TCP. BRSKI relies on Enrollment over Secure Transport (EST) [RFC7030] for the provisioning of the operational domain certificates.

EST-coaps provides a subset of EST functionality and extends EST with BRSKI functions. EST-coaps replaces the invocations of TLS and HTTP by DTLS and CoAP invocations thus enabling EST and BRSKI for CoAP-based low-resource devices.

Although EST-coaps paves the way for the utilization of EST for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with EST-coaps. The specification of EST-coaps is intended to ensure that bootstrapping works for less constrained devices that choose to limit their communications stack to UDP/CoAP. It is up to the network

designer to decide which devices execute the EST protocol and which not.

EST-coaps is designed for use in professional control networks such as Building Control. The autonomic bootstrapping is interesting because it reduces the manual intervention during the commissioning of the network. Typing in passwords is contrary to this wish. Therefore, the HTTP Basic authentication of EST is not supported in EST-coaps.

In the constrained devices context it is very unlikely that full PKI request messages will be used. For that reason, full PKI messages are not supported in EST-coaps.

Because the relatively large EST messages cannot be readily transported over constrained (6LoWPAN, LLN) wireless networks, this document specifies the use of CoAP Block-Wise Transfer ("Block") [RFC7959] to fragment EST messages at the application layer.

Support for Observe CoAP options [RFC7641] with BRSKI is not supported in the current BRSKI/EST message flows and is thus out-of-scope for this discussion. Observe options could be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

1.1. Terminology

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

Many of the concepts in this document are taken over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

The following terms are defined in the BRSKI protocol [I-D.ietf-anima-bootstrapping-keyinfra]: pledge, Join proxy, Join Registrar, and Manufacturer Authorized Signing Authorities (MASA).

2. EST operational differences

Only the differences to EST with respect to operational scenarios are described in this section. EST-coaps server differs from EST server as follows:

- Replacement of TLS by DTLS and HTTP by CoAP, resulting in:

    * DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.

- Only certificate-based client authentication is supported, which results in:

    * The EST-coaps client does not support HTTP Basic authentication (as described in Section 3.2.3 of [RFC7030])

    * The EST-coaps client does not support authentication at the application layer (as described in Section 3.2.3 of [RFC7030]).

- EST-coaps does not support full PKI request messages[RFC5272].

- EST-coaps specifies the BRSKI extensions over CoAP as specified in section 5 of [I-D.ietf-anima-bootstrapping-keyinfra].

3. Conformance to RFC7925 profiles

This section shows how EST-coaps fits into the profiles of low-resource devices as described in [RFC7925]. Within the bootstrap context a Public Key Infrastructure (PKI) is used, where the client is called "pledge", the Registration Authority (RA) is called Join Registrar, which acts at the front-end for the Certificate Authority (CA) and receives voucher feedback from as many Manufacturer Authorized Signing Authorities (MASA) as there are manufacturers. A Join-Proxy is placed between client and RA to receive join requests over a 1-hop unsecured channel and transmitted over the secure network to the EST-server. The EST-server of EST-coaps is placed between proxy and RA or is part of RA.

EST-coaps transports Public keys and certificates. Private keys can be transported as response to a request to a server-side key generation as described in section 4.4 of [RFC7030]. In the bootstrapping context, EST-coaps transport is limited to the EST certificate transport conformant to section 4.4 of [RFC7925]. For BRSKI, outside the profiles of [RFC7925], EST-coaps transports vouchers, which are YANG files specified in [I-D.ietf-anima-voucher].

The mandatory cipher suite for DTLS is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251] which is the mandatory-to-implement cipher suite in CoAP. Additionally the curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. DTLS implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The EST-coaps client MUST be configured with an explicit TA database or at least an implicit TA database from its manufacturer. The authentication of the EST-coaps server by the EST-coaps client is based on Certificate authentication in the DTLS handshake.

The authentication of the EST-coaps client is based on client certificate in the DTLS handshake. This can either be

- DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients;
- DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

4. Protocol Design and Layering

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) 6LoWPAN fragmentation of UDP datagrams. The use of "Block" for the transfer of larger EST messages is specified in Section 4.5. The Figure 1 below shows the layered EST-coaps architecture.

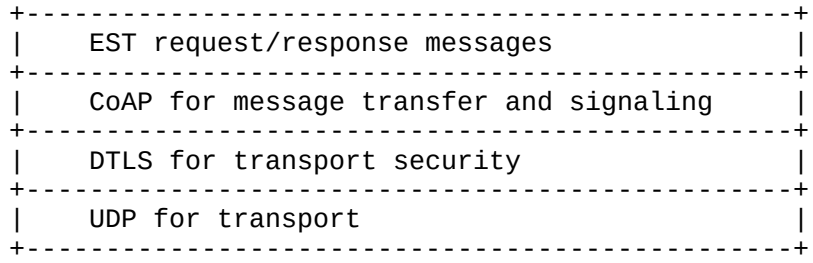


Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design, excluding some aspects that are not relevant for automatic bootstrapping of constrained devices within a professional context. The parts supported by EST-coaps are identified by their message types:

- Simple enroll and reenroll.
- CA certificate retrieval.
- CSR Attributes request messages.

o Server-side key generation messages.

4.1. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and the EST-coaps URI is shorter than the EST URI.

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.est" [RFC6690]. Upon success, the return payload will contain the root resource of the EST resources. It is up to the implementation to choose its root resource, but it is recommended that the value "/est" is used, where possible. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.est

RES: 2.05 Content </est>; rt="core.est"
```

The EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying shorter resource path names:

coaps://www.example.com/est/short-name

Figure 5 in section 3.2.2 of [RFC7030] enumerates the operations and corresponding paths which are supported by EST. Table 1 provides the mapping from the EST and BRSKI URI path to the EST-coaps URI path.

BRSKI	EST	EST-coaps
	/cacerts	/crts
	/simpleenroll	/sen
	/simplereenroll	/sren
	/csrattrs	/att
	/serverkeygen	/skg
/requestvoucher		/rv
/voucherstatus		/vs
/enrollstatus		/es

Table 1

/requestvoucher and /enrollstatus are needed between pledge and Registrar.

4.2. Payload format

The content-format (media type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The media types specified in the HTTP Content-Type header(see section 3.2.2 of [RFC7030]) are in EST-coaps specified by the Content-Format Option (12) of CoAP. The combination of URI path-suffix and content-format used for coap MUST map to an allowed combination of path-suffix and media type as defined for EST. The required content-formats for these request and response messages are defined in Section 8. The CoAP response codes are defined in Section 4.4.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple CBOR byte string is more efficient (30% less payload compared to base64) and well supported by CoAP. Therefore, the content formats specification in Section 8 requires the use of CBOR byte string (h'xxxx' in Diagnostic JSON) for all EST-coaps CoAP payloads.

4.3. Message Bindings

This section describes BRSKI to CoAP message mappings.

All /crts, /sen, /sren, /att, /skg, /rv, /vs, and /es EST-coaps messages expect a response, so they are all CoAP CON messages.

The Ver, TKL, Token, and Message ID values of the CoAP header are not influenced by EST.

CoAP options are used to convey Uri-Host, Uri-Path, Uri-Port, Content-Format and more in CoAP. The CoAP Options are used to communicate the HTTP fields specified in the BRSKI REST messages.

BRSKI URLs are HTTPS based (https://), in CoAP these will be assumed to be transformed to coaps (coaps://)

Appendix A includes some practical examples of EST messages translated to CoAP.

4.4. CoAP response codes

Section 5.9 of [RFC7252] specifies the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request, in EST-coaps the equivalent CoAP response code 2.05 MUST be used. Response code HTTP

202 in EST is mapped to CoAP 2.06 as specified in [I-D.hartke-core-pending]. All other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415 ; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

Appendix A includes some practical examples of HTTP response codes from EST translated to CoAP.

4.5. Message fragmentation

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states "Each DTLS record MUST fit within a single datagram". In order to avoid using IP fragmentation, which is not supported by 6LoWPAN, invokers of the DTLS record layer MUST size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 network SHOULD attempt to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames.

That is not always possible. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certs that amount to large payloads. CoAP [RFC7252]’s section 4.6 describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Also "If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; per [RFC0791], the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload". Thus, even with ECC certs, EST-coaps messages can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919] as explained in section 2 of [RFC7959]. EST-coaps needs to be able to fragment EST messages into multiple DTLS datagrams with each DTLS datagram. Fine-grained fragmentation of EST messages is essential.

To perform fragmentation in CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload of a CoAP flow.

The BLOCK draft defines SZX in the Block1 and block2 option fields. These are used to convey the size of the blocks in the requests or responses.

The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size. As explained in Section 1 of [RFC7959]), blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks.

The Size1 response MAY be parsed by the client as a size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

Examples of fragmented messages are shown in Appendix B.

5. Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the CoAP messages exchanged.

5.1. DTLS

DTLS is one such secure protocol. Within BRSKI and EST when "TLS" is referred to, it is understood that in EST-coaps, security is provided using DTLS instead. No other changes are necessary (all provisional modes etc are the same as for TLS).

CoAP was designed to avoid fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. If fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over a number of records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

EST-coaps does not support full PKI Requests. Consequently, the fullcmc request of section 4.3 of [RFC7030] and response MUST NOT be supported by EST-coaps.

Channel-binding information for linking proof-of-identity with message-based proof-of-possession is optional for EST-coaps. Given that CoAP and DTLS can provide proof of identity for EST-coaps clients and server, simple PKI messages can be used conformant to

section 3.1 of [RFC5272]. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

When proof-of-possession is desired, a set of actions are required regarding the use of tls-connect, described in section 3.5 in [RFC7030] -- Linking Identity and POP Information. The tls-unique information translates to the contents of the first "Finished" message in the TLS handshake between server and client. The client is then supposed to add this "Finished" message as a ChallengePassword to the PKCS#10 to prove that the client is indeed in control of the private key at the time of the TLS session when performing a /simpleenroll, for example. In the case of EST-coaps, the same operations can be performed during the DTLS handshake.

In a constrained CoAP environment, endpoints can’t afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. The DTLS connection SHOULD remain open for persistent EST connections. For example, an EST cacerts request that is followed by a simpleenroll request can use the same authenticated DTLS connection. Given that after a successful enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other.

5.2. 6tisch approach

The 6tisch bootstrapping is targeted to the "imprinting" of the "pledge" with layer 2 keys. The content formats for the transport are being defined and may be expressed in a YANG module.

Instead of using transport security, the 6tisch approach relies on application security provided by OSCOAP [I-D.ietf-core-object-security].

It is suggested that the EST-coaps communication between pledge and registrar, specified in this document, can be freely exchanged with the same communication specified in [I-D.ietf-6tisch-dtsecurity-secure-join] and [I-D.ietf-6tisch-minimal-security].

[EDNOTE: The evolution of this section depends on the directions taken by 6tisch and anima and the possible commonality that will be provided.]

6. Proxying

[EDNOTE: This section to be populated. It will address how proxying can take place by an entity that resides at the edge of the CoAP network, such as the Registrar, and can reach the BRSKI server residing in a traditional "TCP setting". It makes sense to mention the properties that the proxy has to fulfill.]

7. Parameters

[EDNOTE: This section to be populated. It will address transmission parameters for BRSKI described in sections 4.7 and 4.8 of the CoAP draft. BRSKI does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting BRSKI. For example the processing delay of CAs could be less then 2s, but in this case they should send a CoAP ACK every 2s while processing.]

8. IANA Considerations

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the below media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

1.

* application/pkcs7-mime
* Type name: application
* Subtype name: pkcs7-mime
* smime-type: certs-only
* ID: TBD1
* Required parameters: None
* Optional parameters: None
* Encoding considerations: CBOR byte string
* Security considerations: As defined in this specification
* Published specification: [RFC5751]

   * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

2.

   * application/pkcs8

   * Type name: application

   * Subtype name: pkcs8

   * ID: TBD2

   * Required parameters: None

   * Optional parameters: None

   * Encoding considerations: CBOR byte string

   * Security considerations: As defined in this specification

   * Published specification: [RFC5958]

   * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

3.

   * application/csrattrs

   * Type name: application

   * Subtype name: csrattrs

   * ID: TBD3

   * Required parameters: None

   * Optional parameters: None

   * Encoding considerations: CBOR byte string

   * Security considerations: As defined in this specification

   * Published specification: [RFC7030]

   * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

4.

* application/pkcs10
* Type name: application
* Subtype name: pkcs10
* ID: TBD4
* Required parameters: None
* Optional parameters: None
* Encoding considerations: CBOR byte string
* Security considerations: As defined in this specification
* Published specification: [RFC5967]
* Applications that use this media type: ANIMA bootstrap (BRSKI) and EST
*
  + application/pkcs12
  + Type name: application
  + Subtype name: pkcs12
  + ID: TBD5
  + Required parameters: None
  + Optional parameters: None
  + Encoding considerations: CBOR byte string
  + Security considerations: As defined in this specification
  + Published specification: IETF
  + Applications that use this media type: ANIMA bootstrap (BRSKI) and EST
*

+ application/auditnonce

+ Type name: application

+ Subtype name: auditnonce

+ ID: TBD6

+ Required parameters: None

+ Optional parameters: None

+ Encoding considerations: CBOR byte string

+ Security considerations: As defined in this specification

+ Published specification: BRSKI??

+ Applications that use this media type: ANIMA bootstrap (BRSKI)

*

+ application/authorizationvoucher

+ Type name: application

+ Subtype name: authorizationvoucher

+ ID: TBD7

+ Required parameters: None

+ Optional parameters: None

+ Encoding considerations: CBOR byte string

+ Security considerations: As defined in this specification

+ Published specification: BRSKI??

+ Applications that use this media type: ANIMA bootstrap (BRSKI)

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

o rt="core.est" needs registration with IANA.

[EDNOTE: This section will be expanded to include types needed that do not exist in CoAP.]

9. Security Considerations

[EDNOTE: This section to be populated. This document describes an existing protocol moved to CoAP and there should not be additional security concerns added beyond the protocol's or CoAP's specifics security considerations. The security considerations mentioned in EST applies also to EST-coaps. Specifically for server-side key generation, it introduces implications for the endpoints and their private keys, which will be covered here.]

10. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution. They would also like to thank Peter Panburana from Cisco for his feedback on technical details of the solution.

11. Change Log

-01:

Merging of draft-vanderstok-ace-coap-est-00 and draft-pritikin-coap-bootstrap-01

URI and discovery are modified

More text about 6tisch bootstrap including EDHOC and OSCOAP

mapping to DICE IoT profiles

adapted to BRSKI progress

12. References

12.1. Normative References

[I-D.hartke-core-pending]
Stok, P. and K. Hartke, "The 'Pending' Response Code for the Constrained Application Protocol (CoAP)", draft-hartke-core-pending-00 (work in progress), February 2017.

[I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-04 (work in progress), October 2016.

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

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <http://www.rfc-editor.org/info/rfc5272>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <http://www.rfc-editor.org/info/rfc5751>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <http://www.rfc-editor.org/info/rfc5967>.

[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>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <http://www.rfc-editor.org/info/rfc7030>.

[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>.

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

12.2. Informative References

[I-D.ietf-6tisch-dtsecurity-secure-join]
Richardson, M., "6tisch Secure Join protocol", draft-ietf-6tisch-dtsecurity-secure-join-01 (work in progress), February 2017.

[I-D.ietf-6tisch-minimal-security]
Vucinic, M., Simon, J., and K. Pister, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-01 (work in progress), February 2017.

[I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "Voucher and Voucher Revocation Profiles for Bootstrapping Protocols", draft-ietf-anima-voucher-00 (work in progress), January 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-01 (work in progress), December 2016.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-04 (work in progress), October 2016.

[ieee802.15.4]
Institute of Electrical and Electronics Engineers, , "IEEE Standard 802.15.4-2006", 2006.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <http://www.rfc-editor.org/info/rfc4919>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[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>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

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

[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, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <http://www.rfc-editor.org/info/rfc7251>.

[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>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <http://www.rfc-editor.org/info/rfc7925>.

Appendix A. EST messages to EST-coaps

[EDNOTE: This section to be expanded to ensure it covers all BRSKI edge conditions.]

A.1. cacerts

In EST, an HTTPS cacerts message can be

```
GET /.well-known/est/cacerts HTTP/1.1
     User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0
                 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
     Host: 192.0.2.1:8085
     Accept: */*
```

The corresponding secure CoAP request is

```
GET coaps://[192.0.2.1:8085]/est/crts
```

with CoAP fields

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.01 is GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4  (option nr = 4+3=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x9
     Option Value = /est/crts
  Payload = [Empty]
```

A 200 OK response with a cert in EST will then be

```
     200 OK
      Status: 200 OK
      Content-Type: application/pkcs7-mime
      Content-Transfer-Encoding: base64
      Content-Length: 4246 [EDNOTE: this example overflows and would
                            need fragmentation. Choose a better example.
                            Regardless we might need an CoAP option for
                            the content-length ie the CoAP payload?)

      MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExADALBgkqhkiG9w0BBwGgggwMMIIC
      +zCCAeOgAwIBAgIJAJpY3nUZO3qcMA0GCSqGSIb3DQEBBQUAMBsxGTAXBgNVBAMT
      ...
```

The corresponding CoAP response is

2.05 Content (Content-Format: application/pkcs7-mime)
{payload}

with CoAP fields

```
  Ver = 1
  T = 2 (ACK)
  Code = 0x45 (2.05 Content)
  Options
    Option1 (Content-Format)
      Option Delta = 0xC  (option nr = 12)
      Option Length = 0x2
      Option Value = TBD1 (defined in this note)

  Payload = h'123456789ABCDEF...'
```

A.2. enroll / reenroll

[EDNOTE: username/password authentication can be described here but is not a primary focus for BRSKI. It is important for generic EST exchanges but would an endpoint device with sufficient user interface to allow username/password input from an end user be required to use CoAP instead of a full HTTPS exchange?]

[EDNOTE: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]

[EDNOTE: Include CoAP message examples.]

A.3. csrattr

[EDNOTE: Include CoAP message examples.]

A.4. enrollstatus

[EDNOTE: Include CoAP message examples.]

A.5. voucher_status

[EDNOTE: Include CoAP message examples.]

A.6. requestvoucher

[EDNOTE: Include CoAP message examples.]

A.7. requestlog

[EDNOTE: Include CoAP message examples.]

[EDNOTE: More examples can be added, for server-side key generation in CMS envelopes.]

Appendix B. EST-coaps Block message examples

This section provides a detailed example of the messages using DTLS and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is the example value assumed for the DTLS datagram size. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a valid /cacerts exchange over DTLS. . The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 3185 bytes in binary. The CoAP message adds around 10 bytes, the DTLS record 29 bytes. To avoid IP fragmentation, the CoAP block option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 50 packets with a payload of 64 bytes each. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 50 times. The server returns an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP response. The CoAP request-response exchange with block option is shown below. Block option is shown in a decomposed way indicating the kind of Block option (2 in this case because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 means last block), and block size exponent (2**(SZX+4)) separated by slashes. The Length 64 is used with SZX= 2 to avoid IP fragmentation. The CoAP Request is sent with

confirmable (CON) option and the content format of the Response is /application/cacerts.

```
GET [192.0.2.1:8085]/est/crts     -->
              <--   (2:0/1/64) 2.05 Content
    GET URI (2:1/1/64)                          -->
              <--   (2:1/1/64) 2.05 Content
                      |
                      |
                      |
     GET URI (2:49/1/64)                        -->
              <--   (2:49/0/64) 2.05 Content
```

For further detailing the CoAP headers of the first two blocks are written out.

The header of the first GET looks like:

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.1 GET)
  Options
   Option1 (Uri-Host)
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)
     Option Delta = 0x4   (option nr = 3+4=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4    (option nr = 7+4=11)
     Option Length = 0x9
     Option Value = /est/crts
  Payload = [Empty]
```

The header of the first response looks like:

[EDNOTE: The contents of the payload do not need to be written as they are encoded with DTLS into something unreadable.]

```
     Ver = 1
     T = 2 (ACK)
     Code = 0x45 (2.05 Content.)
     Options
       Option1 (Content-Format)
         Option Delta = 0xC  (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x0A (block number = 0, M=1, SZX=2)
     Payload = h'123456789ABCDEF...' (512 bytes)
```

The second Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x45 (2.05 Content.)
     Options
       Option1 (Content-Format)
         Option Delta = 0xC   (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x1
         Option Value = 0x1D (block number = 1, M=1, SZX=2)
     Payload = = h'123456789ABCDEF...' (512 bytes)
```

The 49th and final Block2:

```
     Ver = 1
     T = 2 (means ACK)
     Code = 0x21
     Options
       Option1 (Content-Format)
         Option Delta = 0xC   (option 12)
         Option Length = 0x2
         Option Value = TBD1
       Option2 (Block2)
         Option Delta = 0xB  (option 23 = 12 + 11)
         Option Length = 0x2
         Option Value = 0x312 (block number = 49, M=0, SZX=2)
     Payload = = h'123456789ABCDEF...' (512 bytes)
```

Authors' Addresses

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

J. Zhu
Huawei

March 13, 2017

Offline usage of ACE
draft-zhu-ace-offline-00

Abstract

[I-D.ietf-ace-oauth-authz] defines a framework for both authentication and authorization in constrained Internet of Things (IoT) environments. A constrained node in this framework may have constraints in computational capability, memory storage, lack of user interface, transmission bandwidth and/or power supply. Battery-powered devices are widely used in IoT deployments and they sleep most of their lifetime for battery saving. Hence, they are usually disconnected from other nodes. This draft provides an overview of the disconnection use cases and discusses offline authentication and authorization solutions.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-ace-oauth-authz] defines a framework for both authentication and authorization in constrained Internet of Things (IoT) environments. The framework is based on a set of building blocks including OAuth 2.0 and CoAP.
Figure 1/[I-D.ietf-ace-oauth-authz] describes the basic ACE protocol flow. The diagram is repeated below.

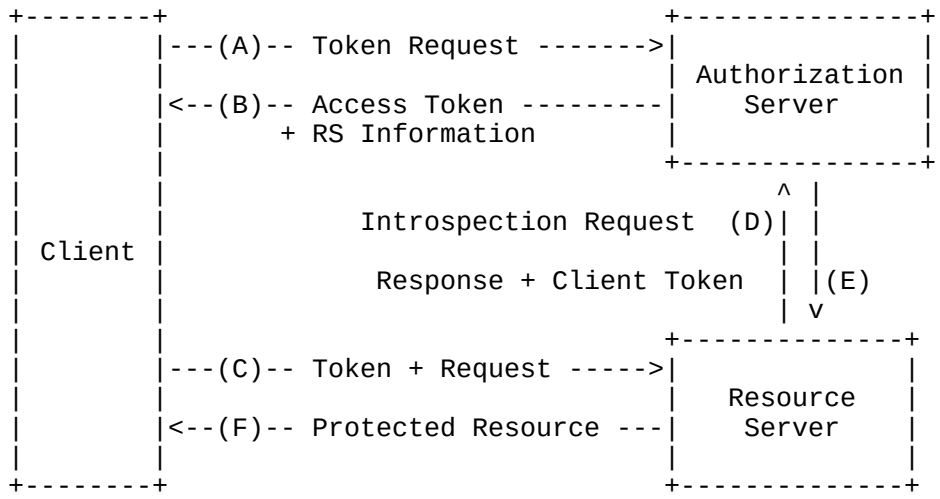


Figure 1: Basic Protocol Flow

(A) The client makes an access token request to the /token endpoint at the Authorization Server (AS).

(B) The AS successfully processes the request from the client, then returns an access token and some RS information.

(C) The client interacts with the resource server (RS) to request access to the protected resource and provides the access token.

(D) The RS may make an introspection request to the /introspect endpoint at the AS to get more information about the access token.

(E) The AS validates the token and returns the most recent parameters associated with it back to the RS.

(F) The RS uses the token information to process the resource access request and returns the protected resources back to the client.

Note: Step D and E are optional steps as the RS can process the access token information locally depending on the deployment configurations.

There may be many constraints for a constrained IoT device such as limited computational capability, memory storage, lack of user interface, transmission bandwidth and/or power supply. According to the [I-D.ietf-ace-actors], either the client or the RS MAY be a constrained node. One critical issue for IoT ecosystems is that more and more constrained devices are battery-powered, e.g. smart water

meters. These battery-powered constrained devices sleep most of their lifetime to save power. What's more, in deployments the underlying network between different nodes may vary from cellular to WLAN even NFC. That means any two nodes of the ACE framework may be disconnected from each other.

As a result of Figure 1, there are 3 different possible disconnection cases between the nodes in the ACE framework:

1. Client-AS disconnection

2. RS-AS disconnection

3. Client-RS disconnection

This document provides an overview of these cases and discusses offline authentication and authorization solutions based on the ACE framework for each of the cases.

The cases discussed in this document utilise the A to F designations from Figure 1 to maintain a relation to the functional steps.

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [I-D.ietf-ace-oauth-authz] and [RFC7252].

3. Cases

This section discusses the disconnection cases including:

1. Client-AS disconnection

2. RS-AS disconnection

3. Client-RS disconnection

Each of the cases may have one or more sub-cases.

For each case there is a brief description at the beginning, and then a possible solution for the disconnection case is discussed.

3.1. Case 1 Client-AS disconnection

In this case we consider the case where the Client is disconnected from the Authorization Server when the Client wants to access a resource on the Resource Server. This usually happens when the network between client and AS goes down, but the client can communicate with the RS via another network.

3.1.1. Sub-case 1 Client instructs the RS to obtain authorization information from AS

This example shows the interaction between a remote controller (Client), a smart television (RS) and a Hub (AS). The remote controller is disconnected from the AS because its WIFI function doesn’t work well. However it can communicate with the smart TV via Bluetooth.

This access procedure involves all the steps shown in Figure 1. In this case, it is assumed that there is a DTLS connection between the client and RS and a separate DTLS connection between the RS and AS.

The client firstly tries to turn on the RS without any authorization information.

C: The client sends a request message to the RS in order to change the state of the switch. However this message does not contain any authorization information.

F: After receiving the request message, the RS verifies it and sends an authorization verification failure response back to the client. The payload of the response MAY contain the AS information in order to instruct the client to obtain an access token from the right address.

Messages C and F is shown in Figure 2.

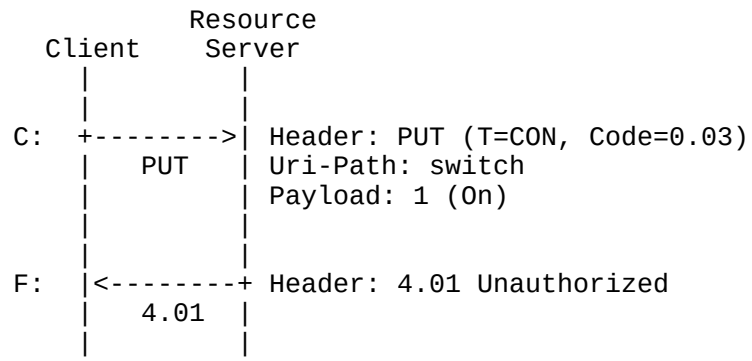


Figure 2: Authorization Failure

After receiving the unauthorized failure message from the RS, the client then tries to request an access token from the AS.

A: The client sends an authorization request to the AS.

B: Because the client is disconnected from the AS, the access token request does not receive a response from the AS and the request times out.

Message A and B are shown in Figure 3.

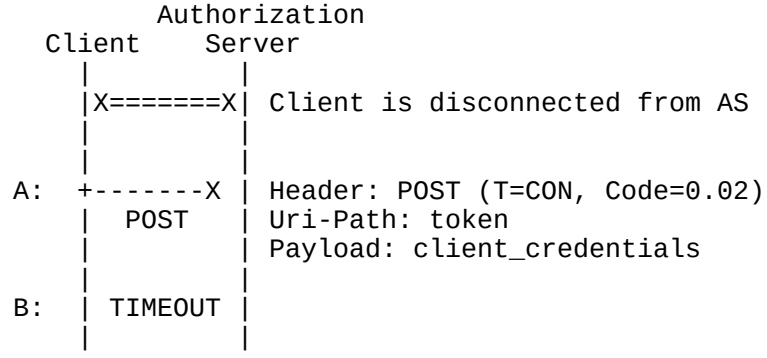


Figure 3: Authorization Timeout

Question: Would it be possible to use the resource server as a proxy to get the authorization information?

3.1.2. Sub-case 2 Introspection Aided Token Validation

In this scenario we consider the same example shown in Section 3.1.1. The difference is that the client has previously (when it could communicate with the AS) received a pre-provisioned long-lived access token before it went offline. The RS uses its online connectivity to validate the access token with the AS.

Note: This is the same use case as the example described in section E.2 of [I-D.ietf-ace-oauth-authz].

3.1.3. Sub-case 3 RS caches authorization information

In this section we consider the same case mentioned in Section 3.1.1.

It is assumed the client can communicate with the AS over a DTLS channel before it goes offline. A DTLS channel is also established between AS and RS as well as a separate channel between the client and RS.

The RS has the capability to cache client authorization information.

Question: Would it be acceptable for the RS to have its cache managed by the client?

3.2. RS-AS disconnection

3.2.1. Sub-case 1: Local Token Validation

In this scenario we consider the case where the resource server is offline, i.e. it is not connected to the AS at the time of the access request. This access procedure involves steps A, B, C, and F of Figure 1.

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

Note: This case is the same as the example described in section E.1 of [I-D.ietf-ace-oauth-authz].

3.3. Client-RS disconnection

In this scenario we consider the case where the client is disconnected from resource server at the time of the access request. For example, both a mobile phone (Client) and a thermostat(RS) are connecting to a same cloud server(AS). The phone has no connection to the thermostat, but the AS should provide a mechanism for the

client to query the temperature remotely. This access procedure involves steps A, B, D, and E of Figure 1 as shown below.

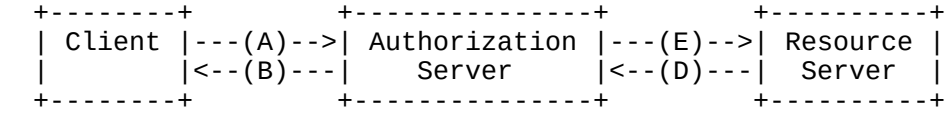


Figure 4: Client-RS disconnection

In this case, it is assumed that a DTLS channel is established between the client & AS, and a separate DTLS connection between the AS & RS as well. The AS SHOULD act as proxy and can forward the resource access request by the client to the RS. The client prior to sending a message to the AS, tried to access the resource directly. However it did not get a successful response due to disconnection between these two nodes. So the client then tries to access the resource via the AS.

Question: Would it be acceptable for the AS to act as a proxy for requests to the RS?

4. Security Considerations

This document addresses authorised access to resources in device disconnection scenarios.

5. IANA Considerations

TBD.

6. Acknowledgements

TBD.

7. Changelog

Initial version

8. References

8.1. Normative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-05 (work in progress), March 2017.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-05 (work in progress), February 2017.

[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

[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>.

Author’s Address

Jintao Zhu
Huawei
P.R.China

Email: jintao.zhu@huawei.com