

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2018

S. Aragon
M. Tiloca
S. Raza
RISE SICS AB
October 29, 2017

IPsec profile of ACE
draft-aragon-ace-ipsec-profile-01

Abstract

This document defines a profile of the ACE framework for authentication and authorization. It uses the IPsec protocol suite and the IKEv2 protocol to ensure secure communication, server authentication and proof-of-possession for a key bound to an OAuth 2.0 access token.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 2, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Methods for Setting Up SA Pairs	4
2.1. The "ipsec" Structure	5
3. Protocol Description	7
3.1. Unauthorized Client to RS	8
3.2. Client to AS	8
3.2.1. Direct Provisioning of SA pairs	9
3.2.2. SA Establishment Based on Symmetric Keys	9
3.2.3. SA Establishment Based on Asymmetric Keys	11
3.3. Client to RS	11
3.3.1. SA Direct Provisioning	12
3.3.2. Authenticated SA Establishment	13
3.4. RS to AS	13
4. Security Considerations	14
4.1. Privacy Considerations	14
5. IANA Considerations	14
5.1. CoAP-IPsec Profile registration	14
5.2. Confirmation Methods registration	15
5.2.1. IPsec field	15
5.2.2. Key Management Protocol field	15
5.3. Key Management Protocol Methods Registry	15
5.3.1. Registration Template	15
5.3.2. Initial Registry Contents	16
6. Acknowledgments	16
7. References	16
7.1. Normative References	17
7.2. Informative References	18
Appendix A. Coexistence of OSCORE and IPsec	18
Appendix B. SA Establishment with EDHOC	20
B.1. Client to AS	20
B.2. Client to RS	21
Authors' Addresses	21

1. Introduction

The IPsec protocol suite [RFC4301] allows communications based on the Constrained Application Protocol (CoAP) [RFC7252] to fulfill a number of security goals at the network layer, i.e. integrity and IP spoofing protection, confidentiality of traffic flows, and message replay protection. In several resource-constrained platforms, this can leverage security operations directly provided by hardware

crypto-modules, including mandatory-to-implement cipher suites defined in [RFC4835].

This document defines a profile of the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz], where a client (C) and a resource server (RS) communicate using CoAP [RFC7252] over IPsec [RFC4301]. In particular, C uses an Access Token released by an Authorization Server (AS) and bound to a key (proof-of-possession key) to authorize its access to RS and its protected resources.

The establishment of an IPsec channel between C and RS provides secure communication, proof-of-possession as well as RS and C mutual authentication. Furthermore, this profile preserves the flexibility of IPsec as to the selection of specific security protocols, i.e. Encapsulating Security Payload (ESP) [RFC4303] and IP Authentication Header (AH) [RFC4302], key management, and modes of operations, i.e. tunnel or transport. Those parameters are specified in the IPsec Security Association (SA) pair established between C and RS. Optionally, the client and the resource server may also use CoAP and IPsec to communicate with the Authorization Server.

This specification supports different key management methods for setting up SA pairs, namely direct provisioning of SA pairs and establishment of SA pairs based on symmetric or asymmetric key authentication. The latter approach relies on the Internet Key Exchange Protocol version 2 (IKEv2) [RFC7296].

1.1. Terminology

In this document, 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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here. These keywords indicate requirement levels for compliant CoAP-IPsec profile implementations.

Readers are expected to be familiar with terminology such as client (C), resource server (RS), authentication server (AS), and endpoint which are defined in [RFC6749] and [I-D.ietf-ace-actors]. It is assumed in this document that a given resource on a specific RS is associated to a unique AS.

The concept of IPsec Security Association (Section 4.1. of [RFC4301]) plays a key role, and this profile uses it extensively. An SA indicates how to secure a one-way communication between two parties. Hence, two SAs are required to be created and coordinated, in order

to secure a two-way communication channel. This document refers to a SA pair as the two IPsec SAs used to protect the two-way communication channel between two IPsec peers.

The SA parameters described in section 4.4.2.1 of [RFC4301] are divided into the following two sets.

- o Network Parameters: the parameters defining the network properties of the IPsec channel, e.g. DSCP filtering;
- o Security Parameters: the parameters defining the security properties of the IPsec channel.

This document refers to SA-C as the SA for securing communication from C to RS, and to SA-RS as the SA for securing communication from RS to C. Thus, a SA pair consists of an SA-C and an SA-RS.

2. Methods for Setting Up SA Pairs

The following key management methods are supported for setting up a SA pair between C and RS.

1. Direct Provisioning (DP). The SA pair is pre-defined by the AS. Then, SA-RS and SA-C are specified in the Access Token Response and in the Access Token issued by the AS.
2. Establishment with symmetric key authentication. A symmetric Pre-Shared Key (PSK) is used to authenticate both parties during the SA pair establishment and is bound to the Access Token as proof-of-possession key. If C is interacting for the first time with the RS, then the AS MUST include a PSK and a unique key identifier in the Access Token Response. Otherwise, C MUST include the unique key identifier pointing at the previously established PSK in the Access Token Request.
3. Establishment with asymmetric key authentication. An asymmetric Raw Public Key (RPK) or Certificate-based Public Key (CPK) is used to authenticate both parties during the SA pair establishment and is bound to the Access Token as proof-of-possession key. If the AS does not know C's asymmetric authentication information, then C MUST include its RPK or CPK in the Access Token Request. Otherwise, C MUST include a key identifier linked to its own RPK or CPK available at the AS.

Every SA MUST include the following Security Parameters.

- o A Security Parameter Index (SPI);

- o IPsec protocol mode: tunnel or transport;
- o Security protocol: AH or ESP;
- o "AH-authentication", "ESP-encryption", "ESP-integrity" or "ESP-combined" algorithm;
- o Source and destination, if tunnel mode is selected;
- o Cryptographic keys;
- o SA lifetime.

As assumed in Section 5.5.2 of [I-D.ietf-ace-oauth-authz], the AS has knowledge of C's and RS's capabilities, and of RS's preferred communications settings. Therefore, the AS MUST set the values of Security Parameters and Network Parameters in the SA pair.

2.1. The "ipsec" Structure

This document defines the "ipsec" structure as a field of the "cnf" parameter of the Access Token and Access Token Response. This structure encodes the Network and Security Parameters of the SA pair as defined in Figure 1. The Network Parameters are not discussed in this specification.

```
ipsec{  
    <Security Parameters>,  
    <Network Parameters>  
}
```

Figure 1: "ipsec" structure overview.

The AS builds the "ipsec" structure as follows:

- o The Security Parameters MUST always include the set of parameters sec_A shown in Figure 2.
- o The Security Parameters MUST include the set of parameters sec_B shown in Figure 3 if the AS uses the Direct Provisioning method.

```
sec_A{
    mode,
    protocol,
    life,
    IP_C,  (if mode == tunnel)
    IP_RS  (if mode == tunnel)
}
```

Figure 2: Set sec_A of Security Parameters

```
sec_B{
    SPI_SA_C,
    SPI_SA_RS,
    alg,
    seed
}
```

Figure 3: Set sec_B of Security Parameters

In sec_A, the IP_C field is the IP address of C, while IP_RS is the IP address of RS. In tunnel mode, the RS MUST use IP_C as the destination address and IP_RS as source address of outgoing IPsec messages. Similarly, C MUST use IP_RS as destination address and IP_C as source address of incoming IPsec messages.

In sec_B, the field "SPI_SA_C" is the SPI of SA-C. Similarly, "SPI_SA_RS" is the SPI of SA-RS. The field "alg" indicates the algorithm used for securing communications over IPsec. The "seed" field MUST reflect the SKEYSEED secret defined in Section 2.14 of [RFC7296]. Thus, C and RS MUST use the same key derivation techniques to generate the necessary SA keys from "seed".

Note that if the Direct Provisioning method is used, the AS cannot guarantee the uniqueness of the "SPI_SA_C" value at the RS and of the "SPI_SA_RS" value at C. In such a case, the AS MUST randomly generate the "SPI_SA_C" value and the "SPI_SA_RS" value, so that the probability of a collision to occur is negligible.

If RS receives an "SPI_SA_C" value which results in a collision, then RS MUST reply to C with an error response, and both C and RS MUST abort the set up of the IPsec channel. In order to overcome this issue, the AS can manage a pool of "SPI_SA_C" reserved values, intended only for use with the Direct Provisioning method. Then, in case of SA termination, the RS asks the AS to set back the identifier of that SA-C as available.

If C receives an "SPI_SA_RS" value which results in a collision, then C sends a second Token Request to the AS, asking for a Token Update.

This Token Request includes also an "ipsec" structure, which contains only the field "SPI_SA_RS" specifying an available value to use. Then, the AS replies with an Access Token and an Access Token Response both updated as to the "SPI_SA_RS" value only.

3. Protocol Description

This profile considers a client C that intends to access a protected resource hosted by a resource server RS. The resource access is authorized through an Access Token issued by the AS as specified in [I-D.ietf-ace-oauth-authz] and indicating that IPsec is used to secure communications between C and RS. In particular, this profile defines how C and RS set up a SA pair, using the key management methods introduced in Section 2.

The protocol is composed of three parts, as shown in Figure 4.

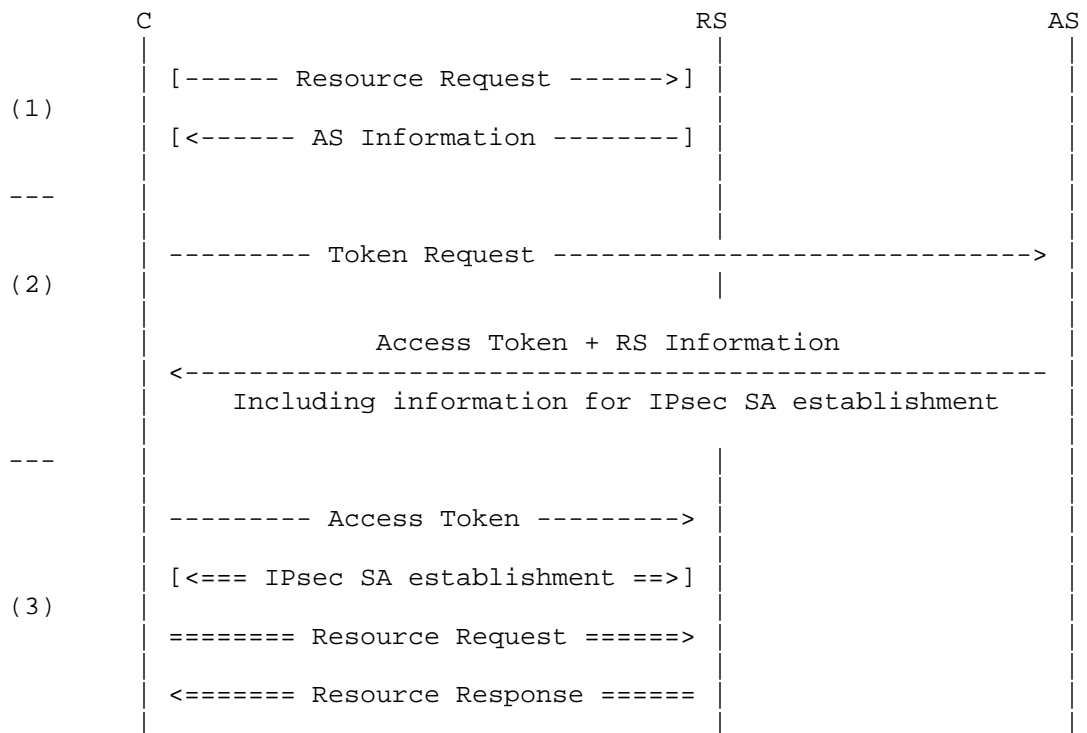


Figure 4: Protocol Overview

3.1. Unauthorized Client to RS

Phase (1) in Figure 4 is OPTIONAL and aims at providing C with the necessary information to contact the AS, in case C does not know AS's address. Through an unauthorized request to RS, C determines which AS is responsible for granting authorization to that particular RS. When doing so, C learns to which address the Access Token Request has to be addressed. The unauthorized request is denied by RS, which sends back to C a response containing the information to contact the AS.

3.2. Client to AS

Phase (2) in Figure 4 starts with C sending the Access Token Request to the /token endpoint at the AS, as specified in Section 5.5.1 of [I-D.ietf-ace-oauth-authz]. Figure 2 and Figure 3 of [I-D.ietf-ace-oauth-authz] provide examples of such request.

If the AS successfully verifies the Access Token Request and C is authorized to access the resource specified in the Token Request, then the AS issues the corresponding Access Token and includes it in a CoAP response with code 2.01 (Created) as specified in Section 5.5.2 of [I-D.ietf-ace-oauth-authz]. The AS can signal that IPsec is REQUIRED to secure communications between C and RS by including the "profile" parameter with the value "coap_ipsec" in the Access Token Response. Together with authorization information, the Access Token also includes the same information for the set up of the IPsec channel included in the Access Token Response. The error response procedures defined in Section 5.5.3 of [I-D.ietf-ace-oauth-authz] are unchanged by this profile.

The information exchanged between C and the AS depends on the specific method used to set up the SA pair (see Section 3.2.1, Section 3.2.2 and Section 3.2.3). Note that, unless Direct Provisioning of SAs is used, C and RS are required to finalize the SA pair set up by running a Key Management Protocol such as IKEv2 (see Section 3.3.2). The AS indicates to use IKEv2 for establishing a SA pair by setting the "kmp" field to "ikev2" in the "cnf" parameter in the Access Token Response.

As specified in Section 5.5 of [I-D.ietf-ace-oauth-authz], the Client and the AS can also use CoAP instead of HTTP to communicate via the /token endpoint. This communication channel MUST be secured.

This section specifies how to use IPsec [RFC4301] to protect the channel between the Client and the AS. The use of IPsec for this communication channel is OPTIONAL in this profile, and other security

protocols MAY be used instead, such as DTLS [RFC6347] and OSCORE [I-D.ietf-core-object-security].

The Client and the AS are either expected to have pre-established a pair of IPsec SA or to have pre-established credentials to authenticate an IKEv2 key exchange. How these credentials are established is out of scope for this profile.

3.2.1. Direct Provisioning of SA pairs

If the AS selects this key management method, it encodes the SA pair in the Access Token and in the Access Token Response as an "ipsec" structure in the "cnf" parameter.

Figure 5 shows an example of an Access Token Response, signaling C to set up an IPsec channel with RS based on the ESP protocol in transport mode.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload : {
  "access_token" : b64'YiksuH&=lGFfg ...
  (remainder of Access Token omitted for brevity)',
  "profile" : "coap_ipsec",
  "expires_in" : "3600",
  "cnf" : {
    "ipsec" : {
      "mode" : "transport",
      "protocol" : "ESP",
      "life" : "3600",
      "SPI_SA_C" : "87615",
      "SPI_SA_RS" : "87616",
      "seed" : b64'+aDg2jjU+eIiOFCa9lObw',
      "alg" : "AES-CCM-16-64-128",
      ... (the Network Parameters are omitted for brevity),
    }
  }
}
```

Figure 5: Example of Access Token Response with DP of SA pair

3.2.2. SA Establishment Based on Symmetric Keys

If the AS selects this key management method, it specifies the following pieces of information in the Access Token Response and in the Access Token:

- o a symmetric key to be used as proof-of-possession key;
- o a key identifier associated to the symmetric key;
- o SA pair's Network Parameters and Security Parameters, as an "ipsec" structure in the "cnf" parameter (see Section 2.1).

If C has previously received a PSK from the AS, then C MUST provide a key identifier of that PSK either directly in the "kid" field of "cnf" parameter or in the "kid" field of the "COSE_Key" object of the Access Token Request. In this case, the AS omits the PSK and its identifier in the Access Token Response.

The AS indicates the use of symmetric cryptography for the key management message exchange in the "kty" field of the "COSE_Key" object, including also the PSK in the "k" field as well as its key identifier in the "kid" field, as shown in Figure 6.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'YiksuH&=lGFfg ...
  (remainder of Access Token omitted for brevity)',
  "profile" : "coap_ipsec",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'6kwi42ec',
      "k" : b64'+pAd48jU+eIiOF23gd=',
    }
    "kmp": "ikev2",
    "ipsec" : {
      "mode" : "tunnel",
      "protocol" : "ESP",
      "life" : "1800",
      "IP_C" : "a.b.c.d2",
      "IP_RS" : "a.b.c.d1",
      ... (the Network Parameters are omitted for brevity),
    }
  }
}
```

Figure 6: Example of Access Token Response with a symmetric key as proof-of-possession key.

3.2.3. SA Establishment Based on Asymmetric Keys

C MUST include its own public key in the Access Token Request, as shown in Figure 7. As an alternative, C MUST provide the key identifier of its own public key, previously shared with the AS.

The AS specifies in the Access Token and in the Access Token Response the SA pair's Network Parameters and Security Parameters, as an "ipsec" structure in the "cnf" parameter (see Section 2.1).

In addition, the AS specifies the RS's public key in the Access Token Response, and the C's public key to be used as proof-of-possession key in the Access Token.

The AS indicates the use of asymmetric cryptography for the key management message exchange in the "kty" field of the "COSE_Key" object, which includes also the RS's public key in the Access Token Response and the C's public key in the Access Token.

```
Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cose+cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "crv" : "P-256",
      "x"   : b64'CaFadPPavdtjRH3YqaTqm0FrFtNV0',
      "y"   : b64'ehekJBwciJdeT6cKieycnk6kg4pHC'
    }
  }
}
```

Figure 7: Example of Access Token Request with an asymmetric key as proof-of-possession key.

3.3. Client to RS

Phase (3) in Figure 4 starts with C posting the Access Token by means of a POST CoAP message to the /authz-info endpoint at RS, as specified in Section 5.7 of [I-D.ietf-ace-oauth-authz]. The processing details of this request, as well as the handling of invalid Access Tokens at RS, are defined in Section 5.7.1 of [I-D.ietf-ace-oauth-authz] and in the rest of this section. The Access Token and Access Token Response specify one of the SA setup

methods defined in Section 2. In particular, C and RS determine the specific SA setup method as follows:

- o In case of Direct Provisioning, the "ipsec" structure is present, while the "COSE_Key" object is not present.
- o If the SA pair set up based on Symmetric Keys through IKEv2 is used, then:
 - * the "COSE_Key" object is present with the "kty" field set to "Symmetric"; and
 - * the "kmp" parameter is set to "ikev2".
- o If the SA pair set up based on Asymmetric Keys through IKEv2 is used, then:
 - * the "COSE_Key" object is present with the "kty" field set to a value that indicates the use of an asymmetric key, e.g. "EC"; and
 - * the "kmp" parameter is set to "ikev2".

If the Direct Provisioning method is used, then C and RS do not perform the SA establishment shown in Figure 4. Otherwise, C and RS perform the key management protocol indicated by the "kmp" parameter (such as IKEv2), in the authentication mode indicated by the "kty" field of the "COSE_key" object.

Regardless the chosen SA setup method and the successful establishment of the IPsec channel, if C holds a valid Access Token but this does not grant access to the requested protected resource, RS MUST send a 4.03 (Forbidden) response. Similarly, if the Access Token does not cover the intended action, RS MUST send a 4.05 (Method Not Allowed) response.

3.3.1. SA Direct Provisioning

Once received a positive Access Token Response from the AS, C derives the necessary IPsec key material from the "seed" field of the "ipsec" structure in the Access Token Response, as discussed in Section 2.1. Similarly, RS performs the same key derivation process upon receiving and successfully verifying the Access Token. After that, RS replies to C with a 2.01 (Created) response, using the IPsec channel specified by the SA pair. Thereafter, Resource Requests and Responses are also sent using the IPsec channel.

3.3.2. Authenticated SA Establishment

If an Authenticated Key Management method is used (see Section 3.2.2 and Section 3.2.3), C and RS MUST run a Key Management Protocol to finalize the establishment of the SA pair and the IPsec channel, i.e. the required keys and algorithms. As shown in Figure 8, the first message `IKE_SA_INIT` of the IKEv2 protocol is used to acknowledge the Access Token submission. Depending on the used authentication method, i.e. symmetric or asymmetric, the proof-of-possession key MUST be used accordingly to authenticate the IKEv2 message exchange as defined in Section 2.15 of [RFC7296]. The rest of the IKEv2 protocol MUST be executed between C and RS as described in Section 2 of [RFC7296], with no further modifications. If IKEv2 is successfully completed, C and RS agree on keys and algorithms to use, and thus the IPsec channel between C and RS is ready to be used.

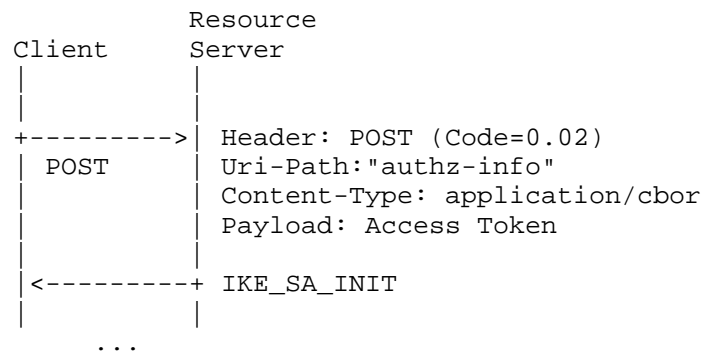


Figure 8: IKEv2 used as Key Management Protocol.

3.4. RS to AS

As specified in Section 5.6 of [I-D.ietf-ace-oauth-authz], the RS and the AS can also use CoAP instead of HTTP to communicate via the /introspect endpoint. This communication channel MUST be secured.

This section specifies how to use IPsec to protect the channel between the RS and the AS. The use of IPsec for this communication channel is OPTIONAL in this profile, and other security protocols MAY be used instead, such as DTLS [RFC6347] and OSCORE [I-D.ietf-core-object-security].

The RS and the AS are either expected to have pre-established a pair of IPsec SA or to have pre-established credentials to authenticate an IKEv2 key exchange. How these credentials are established is out of scope for this profile.

4. Security Considerations

This document inherits the security considerations of [RFC4301], [RFC4302] and [RFC4303]. Furthermore, if IKEv2 is used as key establishment method (see Section 3.3.2), the same considerations discussed in [RFC7296] hold.

4.1. Privacy Considerations

The message exchange in Phase (1) of Figure 4 is unprotected and MAY disclose the relation between the AS, RS and C, as well as network related information, such as IP addresses. Thus RS SHOULD only include the necessary information to contact the AS in the unprotected response.

5. IANA Considerations

This document requires the following IANA considerations:

name	label	CBOR type	value	description
kmp	TBD	bstr	ikev2	Indicates the key management protocol to be used to establish a SA pair
ipsec	TBD	struct		Contains Security and Network Parameters of an SA pair

5.1. CoAP-IPsec Profile registration

- o Profile name: CoAP-IPsec
- o Profile description: ACE Framework profile
- o Profile ID: coap_ipsec
- o Change Controller: IESG
- o Specification Document: This document

5.2. Confirmation Methods registration

5.2.1. IPsec field

- o Confirmation Method Name: "ipsec"
- o Confirmation Method Value: TBD
- o Confirmation Method Description: A structure containing the corresponding information of an IPsec Security Association Pair.
- o Change Controller: IESG
- o Specification Document: This document

5.2.2. Key Management Protocol field

- o Confirmation Method Name: "kmp"
- o Confirmation Method Value: TBD
- o Confirmation Method Description: Key management protocol.
- o Change Controller: IESG
- o Specification Document: This document

5.3. Key Management Protocol Methods Registry

This specification establishes the IANA "Key Management Protocol Methods" registry for the "kmp" member values. The registry records the confirmation method member and a reference to the spec that defines it.

5.3.1. Registration Template

Key Management Protocol Method Name:

The name requested (e.g. "ikev2"). 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.

Key Management Protocol 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.

Key Management Protocol 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.

5.3.2. Initial Registry Contents

- o Key Management Protocol Method Name: "ikev2"
- o Key Management Protocol Method Value: TBD
- o Key Management Protocol Method Description: Defines IKEv2 as key management protocol.
- o Change Controller: IESG
- o Specification Document: this document

6. Acknowledgments

The authors sincerely thank Max Maass for his comments and feedback.

The authors gratefully acknowledge the EIT-Digital Master School for partially funding this work.

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-08 (work in progress), October 2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <<https://www.rfc-editor.org/info/rfc4302>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4835] Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, DOI 10.17487/RFC4835, April 2007, <<https://www.rfc-editor.org/info/rfc4835>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

7.2. 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-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.
- [I-D.seitz-ace-oscoap-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-seitz-ace-oscoap-profile-06 (work in progress), October 2017.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-07 (work in progress), July 2017.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

Appendix A. Coexistence of OSCORE and IPsec

Object Security of Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] is a data object based security protocol that protects CoAP messages end-to-end while allowing proxy operations. It encloses unprotected CoAP messages, and selected CoAP options and headers fields into a CBOR Object Signing and Encryption (COSE) object [RFC8152]. This section describes a scenario where communications between C and RS are secured by means of OSCORE and IPsec. Figure 9 depicts a scenario where a Client needs to access a

Resource Server which is behind an untrusted CoAP-Proxy. This scenario requires that:

1. the Proxy has access to the selected CoAP options to perform management and support operations;
2. the integrity of messages and their IP headers can be verified by the Resource Server;
3. the confidentiality of the Resource Server address and CoAP request has to be guaranteed between the Client and the Proxy.

The first requirement is addressed by means of an OSCORE channel between the Client and the Resource Server established as described in [I-D.seitz-ace-oscoap-profile]), by marking as Class E the sensitive fields of the CoAP payload as defined in [I-D.ietf-core-object-security].

To address the second requirement, a SA pair between the Client and the Resource Server is established, as specified in Section 3, by using the IPsec AH protocol in transport mode. Finally, the third requirement is fulfilled by means of a SA pair between the Client and the CoAP-Proxy, as specified in Section 3, by using the IPsec ESP protocol in tunnel mode.

This profile can be used to establish the necessary SA pairs. After that, C can request a token update to the AS, in order to establish an OSCORE security context with RS, as specified in Section 2.2 of [I-D.seitz-ace-oscoap-profile].

Figure 9 overviews the involved secure communication channels. Logical links such as the SA pair shared between the Client and the Proxy are represented by dotted lines. IPsec traffic is depicted with double-dashed lines, and an example of the packets going through these links is represented with numbers, e.g. (1). The destination address included in the IP headers is also specified, e.g. "IP:P" indicates the Proxy's address as destination address. The source address of the IP header is omitted, since all the IP packets have the Client's address as source address.

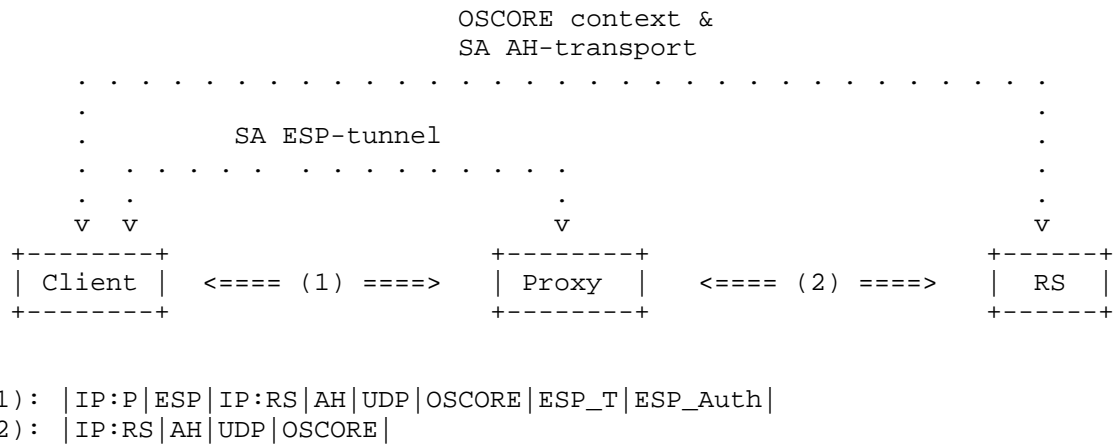


Figure 9: OSCORE and IPsec - Scenario overview

Appendix B. SA Establishment with EDHOC

As discussed in Appendix A, securing communications between C and RS with both OSCORE and IPsec makes it possible to fulfill a number of additional security requirements. An OSCORE security context between C and RS can be established using Ephemeral Diffie-Hellman Over COSE (EDHOC) as defined in Appendix C.2 of [I-D.selander-ace-cose-ecdhe] and according to [I-D.seitz-ace-oscoap-profile]. This section proposes a method to establish also IPsec SA pairs by means of EDHOC. This makes it possible for constrained devices running the scenario described in Appendix A to rely solely on EDHOC for establishing both OSCORE contexts and IPsec SA pairs, thus avoiding to include the implementation of IKEv2 as further key management protocol.

In particular, C and RS can refer to the SA Authenticated Establishment methods described in this specification, and then use EDHOC to finalize the SA pair, i.e. by deriving the encryption and authentication keys for the security protocols specified in the SA pair. This is possible thanks to IPsec's independence from specific key management protocols. In addition, the same security consideration discussed in [I-D.selander-ace-cose-ecdhe] hold.

The AS, C and RS refer to the same protocol shown in Figure 4, with the following changes.

B.1. Client to AS

The AS specifies the fields "alg", "SPI_SA_C" and "SPI_SA_RS" of the "ipsec" structure in the Access Token and in the Access Token Response, in addition to the pieces of information defined in

Section 3.2.2 or Section 3.2.3, in case the proof-of-possession key is symmetric or asymmetric, respectively.

The AS signals that EDHOC MUST be used, by setting the "kmp" field to "edhoc" in the Access Token and the Access Token Response. Then, C and RS MUST perform EDHOC as described in Section 4 or Section 5 of [I-D.selander-ace-cose-ecdhe], in case the proof-of-possession key is asymmetric or symmetric, respectively.

B.2. Client to RS

Figure 10 shows how EDHOC message_1 is sent through a POST Access Token Request to the /authz-info at the RS. The RS SHALL process the Access Token according to [I-D.ietf-ace-oauth-authz], and, if valid, continue with the EDHOC protocol as defined in Appendix C.1 of [I-D.selander-ace-cose-ecdhe]. Otherwise, RS aborts EDHOC and responds with an error code as specified in [I-D.ietf-ace-oauth-authz]. At the end of the EDHOC protocol, C and RS MUST derive an IPsec seed from the EDHOC shared secret. The seed is derived as specified in Section 3.2 of [I-D.selander-ace-cose-ecdhe], with other=exchange_hash, AlgorithmID="EDHOC IKE seed" and keyDataLength equal to the key length of the SKEYSEED secret defined in Section 2.14 of [RFC7296]. After that, the derived seed is written in the "seed" field of the "ipsec" structure, and accordingly used to derive IPsec key material as described in Section 2.1.

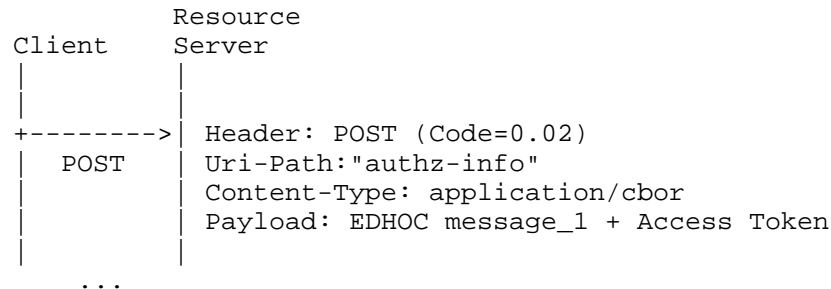


Figure 10: EDHOC used as Key Management Protocol

Authors' Addresses

Santiago Aragon
RISE SICS AB
Isafjordsgatan 22
Kista SE-164 29
Sweden

Email: santiago.aragon@stud.tu-darmstadt.de

Marco Tiloca
RISE SICS AB
Isafjordsgatan 22
Kista SE-164 29
Sweden

Phone: +46 70 604 65 01
Email: marco.tiloca@ri.se

Shahid Raza
RISE SICS AB
Isafjordsgatan 22
Kista SE-164 29
Sweden

Phone: +46 76 883 17 97
Email: shahid.raza@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

M. Jones
Microsoft
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
ARM Ltd.
March 19, 2018

CBOR Web Token (CWT)
draft-ietf-ace-cbor-web-token-15

Abstract

CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. 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. CWT is derived from JSON Web Token (JWT) but uses CBOR rather than JSON.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 20, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. CBOR Related Terminology	3
2. Terminology	3
3. Claims	4
3.1. Registered Claims	5
3.1.1. iss (Issuer) Claim	5
3.1.2. sub (Subject) Claim	5
3.1.3. aud (Audience) Claim	5
3.1.4. exp (Expiration Time) Claim	5
3.1.5. nbf (Not Before) Claim	5
3.1.6. iat (Issued At) Claim	6
3.1.7. cti (CWT ID) Claim	6
4. Summary of the claim names, keys, and value types	6
5. CBOR Tags and Claim Values	6
6. CWT CBOR Tag	6
7. Creating and Validating CWTs	7
7.1. Creating a CWT	7
7.2. Validating a CWT	8
8. Security Considerations	9
9. IANA Considerations	10
9.1. CBOR Web Token (CWT) Claims Registry	10
9.1.1. Registration Template	11
9.1.2. Initial Registry Contents	11
9.2. Media Type Registration	13
9.2.1. Registry Contents	13
9.3. CoAP Content-Formats Registration	14
9.3.1. Registry Contents	14
9.4. CBOR Tag registration	14
9.4.1. Registry Contents	14
10. References	14
10.1. Normative References	14
10.2. Informative References	15
Appendix A. Examples	16
A.1. Example CWT Claims Set	16
A.2. Example keys	16
A.2.1. 128-bit Symmetric Key	17

A.2.2. 256-bit Symmetric Key	17
A.2.3. ECDSA P-256 256-bit COSE Key	17
A.3. Example Signed CWT	18
A.4. Example MACed CWT	19
A.5. Example Encrypted CWT	20
A.6. Example Nested CWT	21
A.7. Example MACed CWT with a floating-point value	22
Appendix B. Acknowledgements	23
Appendix C. Document History	23
Authors' Addresses	27

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) [RFC8152] specification is used.

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

1.1. CBOR Related Terminology

In JSON, maps are called objects and only have one kind of map key: a string. CBOR uses strings, negative integers, and unsigned integers as map keys. The integers are used for compactness of encoding and easy comparison. The inclusion of strings allows for an additional range of short encoded values to be used.

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 BCP

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document reuses terminology from JWT [RFC7519] and COSE [RFC8152].

StringOrURI

The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate

The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead of a JSON number. The encoding is modified so that the leading tag 1 (epoch-based date/time) MUST be omitted.

Claim Name

The human-readable name used to identify a claim.

Claim Key

The CBOR map key used to identify a claim.

Claim Value

The CBOR map value representing the value of the claim.

CWT Claims Set

The 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 Claim Keys are represented using integers or text strings. Section 4 summarizes all keys used to identify the claims defined in this document.

3.1. Registered Claims

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 and processing rules as the "iss" claim defined in Section 4.1.1 of [RFC7519], except that the value is a StringOrURI, as defined in Section 2 of this specification. The Claim Key 1 is used to identify this claim.

3.1.2. sub (Subject) Claim

The "sub" (subject) claim has the same meaning and processing rules as the "sub" claim defined in Section 4.1.2 of [RFC7519], except that the value is a StringOrURI, as defined in Section 2 of this specification. The Claim Key 2 is used to identify this claim.

3.1.3. aud (Audience) Claim

The "aud" (audience) claim has the same meaning and processing rules as the "aud" claim defined in Section 4.1.3 of [RFC7519], except that the value of the audience claim is a StringOrURI when it is not an array or each of the audience array element values is a StringOrURI when the audience claim value is an array. (StringOrURI is defined in Section 2 of this specification.) The Claim Key 3 is used to identify this claim.

3.1.4. exp (Expiration Time) Claim

The "exp" (expiration time) claim has the same meaning and processing rules as the "exp" claim defined in Section 4.1.4 of [RFC7519], except that the value is a NumericDate, as defined in Section 2 of this specification. The Claim Key 4 is used to identify this claim.

3.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim has the same meaning and processing rules as the "nbf" claim defined in Section 4.1.5 of [RFC7519], except that the value is a NumericDate, as defined in Section 2 of this specification. The Claim Key 5 is used to identify this claim.

3.1.6. iat (Issued At) Claim

The "iat" (issued at) claim has the same meaning and processing rules as the "iat" claim defined in Section 4.1.6 of [RFC7519], except that the value is a `NumericDate`, as defined in Section 2 of this specification. The Claim Key 6 is used to identify this claim.

3.1.7. cti (CWT ID) Claim

The "cti" (CWT ID) claim has the same meaning and processing rules as the "jti" claim defined in Section 4.1.7 of [RFC7519], except that the value is a byte string. The Claim Key 7 is used to identify this claim.

4. Summary of the claim names, keys, and value types

Name	Key	Value type
iss	1	text string
sub	2	text string
aud	3	text string
exp	4	integer or floating-point number
nbf	5	integer or floating-point number
iat	6	integer or floating-point number
cti	7	byte string

Table 1: Summary of the claim names, keys, and value types

5. CBOR Tags and Claim Values

The claim values defined in this specification MUST NOT be prefixed with any CBOR tag. For instance, while CBOR tag 1 (epoch-based date/time) 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, this does not prohibit future claim definitions from requiring the use of CBOR tags for those specific claims.

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 1: 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 [RFC8152] 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 [RFC8152] 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 [RFC8152] 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 [RFC8152] 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 tagged COSE_Sign/COSE_Sign1, COSE_Mac/COSE_Mac0, or COSE_Encrypt/COSE_Encrypt0, and return to Step 3.
 6. If needed by the application, prepend the COSE object with the appropriate COSE CBOR tag to indicate the type of the COSE object. If needed by the application, prepend the COSE object with 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 use it to determine the type of the CWT, COSE_Sign/COSE_Sign1, COSE_Mac/COSE_Mac0, or COSE_Encrypt/COSE_Encrypt0. If the object does not have a COSE CBOR tag, the COSE message type is determined from the application context.
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. 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 [RFC8152] 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 [RFC8152] 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 [RFC8152] Section 5 (Encryption Objects) for validating a COSE_Encrypt/COSE_Encrypt0 object. Let the Message be the resulting plaintext.
6. If the Message begins with a COSE CBOR tag, then the Message is a CWT that was the subject of nested signing, MACing, or encryption operations. In this case, return to Step 1, using the Message as the CWT.
 7. Verify that the Message is a valid CBOR map; let the CWT Claims Set be this CBOR map.

8. Security Considerations

The security of the CWT relies upon 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.

While syntactically the signing and encryption operations for Nested CWTs may be applied in any order, if both signing and encryption are necessary, normally producers should sign the message and then encrypt the result (thus encrypting the signature). This prevents attacks in which the signature is stripped, leaving just an encrypted message, as well as providing privacy for the signer. Furthermore, signatures over encrypted text are not considered valid in many jurisdictions.

9. IANA Considerations

9.1. CBOR Web Token (CWT) Claims Registry

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

Registration requests are evaluated using the criteria described in the Claim Key instructions in the registration template below after a three-week review period on the `cwt-reg-review@ietf.org` mailing list, 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. [[Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: `cwt-reg-review@ietf.org`.]]

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to register claim: example"). Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the `iesg@ietf.org` mailing list) for resolution.

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. Registrations for the limited set of values between -256 and 255 and strings of length 1 are to be restricted to claims with general applicability.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification in order to enable broadly informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

Since a high degree of overlap is expected between the contents of the "CBOR Web Token (CWT) Claims" registry and the "JSON Web Token Claims" registry, overlap in the corresponding pools of Designated Experts would be useful to help ensure that an appropriate level of coordination between the registries is maintained.

9.1.1.1. 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".

Claim Key:

CBOR map key for the claim. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 and strings of length 1 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values greater than 65535 and strings of length greater than 2 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Claim Value Type(s):

CBOR types that can be used for the claim value.

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

- o Claim Name: (RESERVED)
- o Claim Description: This registration reserves the key value 0.
- o JWT Claim Name: N/A
- o Claim Key: 0
- o Claim Value Type(s): N/A
- o Change Controller: IESG
- o Specification Document(s): [[this specification]]

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

- o Claim Name: "sub"
- o Claim Description: Subject
- o JWT Claim Name: "sub"
- o Claim Key: 2
- o Claim Value Type(s): text string
- o Change Controller: IESG
- o Specification Document(s): Section 3.1.2 of [[this specification
]]

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

- o Claim Name: "exp"
- o Claim Description: Expiration Time
- o JWT Claim Name: "exp"
- o Claim Key: 4
- o Claim Value Type(s): integer or floating-point number
- o Change Controller: IESG
- o Specification Document(s): Section 3.1.4 of [[this specification
]]

- o Claim Name: "nbf"
- o Claim Description: Not Before
- o JWT Claim Name: "nbf"
- o Claim Key: 5
- o Claim Value Type(s): integer or floating-point number
- o Change Controller: IESG
- o Specification Document(s): Section 3.1.5 of [[this specification
]]

- o Claim Name: "iat"
- o Claim Description: Issued At
- o JWT Claim Name: "iat"

- o Claim Key: 6
- o Claim Value Type(s): integer or floating-point number
- o Change Controller: IESG
- o Specification Document(s): Section 3.1.6 of [[this specification]]
- o Claim Name: "cti"
- o Claim Description: CWT ID
- o JWT Claim Name: "jti"
- o Claim Key: 7
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o 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

- o Type name: application
- o Subtype name: cwt
- o Required parameters: N/A
- o Optional parameters: N/A
- o Encoding considerations: binary
- o Security considerations: See the Security Considerations section of [[this specification]]
- o Interoperability considerations: N/A
- o Published specification: [[this specification]]
- o Applications that use this media type: IoT applications sending security tokens over HTTP(S), CoAP(S), and other transports.
- o Fragment identifier considerations: N/A
- o Additional information:
 - Magic number(s): N/A
 - File extension(s): N/A
 - Macintosh file type code(s): N/A
- o Person & email address to contact for further information: IESG, iesg@ietf.org
- o Intended usage: COMMON
- o Restrictions on usage: none
- o Author: Michael B. Jones, mbj@microsoft.com
- o Change controller: IESG
- o 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

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

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

10. References

10.1. Normative References

[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.MediaTypees]
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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [IANA.JWT.Claims] IANA, "JSON Web Token Claims", <<http://www.iana.org/assignments/jwt>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://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, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<https://www.rfc-editor.org/info/rfc7516>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

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].

Where a byte string is to carry an embedded CBOR-encoded item, the diagnostic notation for this CBOR data item can be enclosed in '<<' and '>>' to notate the byte string resulting from encoding the data item, e.g., h'63666F6F' translates to <<"foo">>.

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 byte string.

```
a70175636f61703a2f2f61732e6578616d706c652e636f6d02656572696b7703
7818636f61703a2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0
051a5610d9f0061a5610d9f007420b71
```

Figure 2: 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 3: 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

a42050231f4c4d4d3051fdc2ec0a3851d5b3830104024c53796d6d6574726963
313238030a

Figure 4: 128-bit symmetric COSE_Key as hex string

```
{  
  / k /   -1: h'231f4c4d4d3051fdc2ec0a3851d5b383'  
  / kty /   1: 4 / Symmetric /,  
  / kid /   2: h'53796d6d6574726963313238' / 'Symmetric128' /,  
  / alg /   3: 10 / AES-CCM-16-64-128 /  
}
```

Figure 5: 128-bit symmetric COSE_Key in CBOR diagnostic notation

A.2.2. 256-bit Symmetric Key

a4205820403697de87af64611c1d32a05dab0fe1fcb715a86ab435f1ec99192d
795693880104024c53796d6d6574726963323536030a

Figure 6: 256-bit symmetric COSE_Key as hex string

```
{  
  / k /   -1: h'403697de87af64611c1d32a05dab0fe1fcb715a86ab435f1  
              ec99192d79569388'  
  / kty /   1: 4 / Symmetric /,  
  / kid /   4: h'53796d6d6574726963323536' / 'Symmetric256' /,  
  / alg /   3: 4 / HMAC 256/64 /  
}
```

Figure 7: 256-bit symmetric COSE_Key in CBOR diagnostic notation

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

a72358206c1382765aec5358f117733d281c1c7bdc39884d04a45a1e6c67c858
bc206c1922582060f7f1a780d8a783bfb7a2dd6b2796e8128dbbcef9d3d168db
9529971a36e7b9215820143329cce7868e416927599cf65a34f3ce2ffda55a7e
ca69ed8919a394d42f0f2001010202524173796d6d6574726963454344534132
35360326

Figure 8: 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 /,
  / kid /  2: h'4173796d6d657472696345434453413
           23536' / 'AsymmetricECDSA256' /,
  / alg /  3: -7 / ECDSA 256 /
}

```

Figure 9: 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.

```

d28443a10126a104524173796d6d657472696345434453413235365850a701756
36f61703a2f2f61732e6578616d706c652e636f6d02656572696b77037818636f
61703a2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0051a5610d
9f0061a5610d9f007420b7158405427c1ff28d23fbad1f29c4c7c6a555e601d6f
a29f9179bc3d7438bacaca5acd08c8d4d4f96131680c429a01f85951ecee743a5
2b9b63632c57209120e1c9e30

```

Figure 10: Signed CWT as hex string


```

18(
  [
    / protected / << {
      / alg / 1: -7 / ECDSA 256 /
    } >>,
    / unprotected / {
      / kid / 4: h'4173796d6d657472696345434453413
        23536' / 'AsymmetricECDSA256' /
    },
    / payload / << {
      / 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'5427c1ff28d23fbad1f29c4c7c6a555e601d6fa29f
      9179bc3d7438bacaca5acd08c8d4d4f96131680c42
      9a01f85951ecee743a52b9b63632c57209120e1c9e
      30'
  ]
)

```

Figure 11: Signed CWT in CBOR diagnostic notation

A.4. Example MACed CWT

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

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.

```

d83dd18443a10104a1044c53796d6d65747269633235365850a70175636f6170
3a2f2f61732e6578616d706c652e636f6d02656572696b77037818636f61703a
2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0051a5610d9f006
1a5610d9f007420b7148093101ef6d789200

```

Figure 12: MACed CWT with CWT tag as hex string

```

61(
  17(
    [
      / protected / << {
        / alg / 1: 4 / HMAC-256-64 /
      } >>,
      / unprotected / {
        / kid / 4: h'53796d6d6574726963323536' / 'Symmetric256' /
      },
      / payload / << {
        / 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 13: 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.

```

d08343a1010aa2044c53796d6d6574726963313238054d99a0d7846e762c49ff
e8a63e0b5858b918a11fd81e438b7f973d9e2e119bcb22424ba0f38a80f27562
f400ee1d0d6c0fdb559c02421fd384fc2ebe22d7071378b0ea7428fff157444d
45f7e6afcdalaae5f6495830c58627087fc5b4974f319a8707a635dd643b

```

Figure 14: Encrypted CWT as hex string

```

16(
  [
    / protected / << {
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963313238' / 'Symmetric128' /,
      / iv / 5: h'99a0d7846e762c49ffe8a63e0b'
    },
    / ciphertext / h'b918a11fd81e438b7f973d9e2e119bcb22424ba0f38
                    a80f27562f400eeld0d6c0fdb559c02421fd384fc2e
                    be22d7071378b0ea7428fff157444d45f7e6afcdala
                    ae5f6495830c58627087fc5b4974f319a8707a635dd
                    643b'
  ]
)

```

Figure 15: 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.

```

d08343a1010aa2044c53796d6d6574726963313238054d4a0694c0e69ee6b595
6655c7b258b7f6b0914f993de822cc47e5e57a188d7960b528a747446fe12f0e
7de05650dec74724366763f167a29c002dfd15b34d8993391cf49bc91127f545
dba8703d66f5b7f1ae91237503d371e6333df9708d78c4fb8a8386c8ff09dc49
af768b23179deab78d96490a66d5724fb33900c60799d9872fac6da3bdb89043
d67c2a05414ce331b5b8f1ed8ff7138f45905db2c4d5bc8045ab372bff142631
610a7e0f677b7e9b0bc73adefdc6e16d9d5d284c616abeab5d8c291ce0

```

Figure 16: Signed and Encrypted CWT as hex string

```

16(
  [
    / protected / << {
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963313238' / 'Symmetric128' /,
      / iv / 5: h'4a0694c0e69ee6b5956655c7b2'
    },
    / ciphertext / h'f6b0914f993de822cc47e5e57a188d7960b528a7474
      46fe12f0e7de05650dec74724366763f167a29c002d
      fd15b34d8993391cf49bc91127f545dba8703d66f5b
      7f1ae91237503d371e6333df9708d78c4fb8a8386c8
      ff09dc49af768b23179deab78d96490a66d5724fb33
      900c60799d9872fac6da3bdb89043d67c2a05414ce3
      31b5b8f1ed8ff7138f45905db2c4d5bc8045ab372bf
      f142631610a7e0f677b7e9b0bc73adefdcce16d9d5d
      284c616abeab5d8c291ce0'
  ]
)

```

Figure 17: Signed and Encrypted CWT in CBOR diagnostic notation

A.7. Example MACed CWT with a floating-point value

This section shows a MACed CWT with a single recipient and a simple CWT Claims Set. The CWT Claims Set with a floating-point 'iat' value.

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.

```

dl8443a10104a1044c53796d6d65747269633235364ba106fb41d584367c2000
0048b8816f34c0542892

```

Figure 18: MACed CWT with a floating-point value as hex string

```

17(
  [
    / protected / << {
      / alg / 1: 4 / HMAC-256-64 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963323536' / 'Symmetric256' /,
    },
    / payload / << {
      / iat / 6: 1443944944.5
    } >>,
    / tag / h'b8816f34c0542892'
  ]
)

```

Figure 19: MACed CWT with a floating-point value 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. It also incorporates suggestions made by many people, including Carsten Bormann, Alissa Cooper, Esko Dijk, Benjamin Kaduk, Warren Kumari, Carlos Martinez, Alexey Melnikov, Kathleen Moriarty, Eric Rescorla, Dan Romascanu, Adam Roach, Kyle Rose, Jim Schaad, Ludwig Seitz, and Goeran Selander.

[[RFC Editor: Is it possible to preserve the non-ASCII spellings of the names Erik Wahlstroem and Goeran Selander in the final specification?]]

Appendix C. Document History

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

-15

- o Added section references when the terms "NumericDate" and "StringOrURI" are used, as suggested by Adam Roach.

-14

- o Cleaned up the descriptions of the numeric ranges of claim keys being registered in the registration template for the "CBOR Web Token (CWT) Claims" registry, as suggested by Adam Roach.

- o Clarified the relationships between the JWT and CWT "NumericDate" and "StringOrURI" terms, as suggested by Adam Roach.
- o Eliminated unnecessary uses of the word "type", as suggested by Adam Roach.
- o Added the text "IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list" from RFC 7519, as suggested by Amanda Baber of IANA, which is also intended to address Alexey Melnikov's comment.
- o Removed a superfluous comma, as suggested by Warren Kumari.
- o Acknowledged additional reviewers.

-13

- o Clarified the registration criteria applied to different ranges of Claim Key values, as suggested by Kathleen Moriarty and Dan Romascanu.
- o No longer describe the syntax of CWT claims as being the same as that of the corresponding JWT claims, as suggested by Kyle Rose.
- o Added guidance about the selection of the Designated Experts, as suggested by Benjamin Kaduk.
- o Acknowledged additional reviewers.

-12

- o Updated the RFC 5226 reference to RFC 8126.
- o Made the IANA registration criteria consistent across sections.
- o Stated that registrations for the limited set of values between -256 and 255 and strings of length 1 are to be restricted to claims with general applicability.
- o Changed the "Reference" field name to "Description of Semantics" in the CBOR Tag registration request.
- o Asked the RFC Editor whether it is possible to preserve the non-ASCII spellings of the names Erik Wahlstroem and Goeran Selander in the final specification.

-11

- o Corrected the "iv" value in the signed and encrypted CWT example.
- o Mention CoAP in the "application/cwt" media type registration.
- o Changed references of the form "Section 4.1.1 of JWT <xref target="RFC7519"/>" to "Section 4.1.1 of <xref target="RFC7519"/>" so that rfcmarkup will generate correct external section reference links.
- o Updated Acknowledgements.

-10

- o Clarified that the audience claim value can be a single audience value or an array of audience values, just as is the case for the JWT "aud" claim.
- o Clarified the nested CWT description.
- o Changed uses of "binary string" to "byte string".

-09

- o Added key ID values to the examples.
- o Key values for the examples are now represented in COSE_Key format using CBOR diagnostic notation.

-08

- o Updated the diagnostic notation for embedded objects in the examples, addressing feedback by Carsten Bormann.

-07

- o Updated examples for signing and encryption. Signatures are now deterministic as recommended by COSE specification.

-06

- o Addressed review comments by Carsten Bormann and Jim Schaad. All changes were editorial in nature.

-05

- o Addressed working group last call comments with the following changes:

- o Say that CWT is derived from JWT, rather than CWT is a profile of JWT.
- o Used CBOR type names in descriptions, rather than major/minor type numbers.
- o Clarified the NumericDate and StringOrURI descriptions.
- o Changed to allow CWT claim names to use values of any legal CBOR map key type.
- o Changed to use the CWT tag to identify nested CWTs instead of the CWT content type.
- o Added an example using a floating-point date value.
- o Acknowledged reviewers.

-04

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

-03

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

-02

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

-01

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

-00

- o 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
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2020

M. Jones
Microsoft
L. Seitz
RISE SICS
G. Selander
Ericsson AB
S. Erdtman
Spotify
H. Tschofenig
Arm Ltd.
October 31, 2019

Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)
draft-ietf-ace-cwt-proof-of-possession-11

Abstract

This specification describes how to declare in a CBOR Web Token (CWT) (which is defined by RFC 8392) that the presenter of the CWT possesses a particular proof-of-possession key. Being able to prove possession of a key is also sometimes described as being the holder-of-key. This specification provides equivalent functionality to "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" (RFC 7800) but using Concise Binary Object Representation (CBOR) and CWTs rather than JavaScript Object Notation (JSON) and JSON Web Tokens (JWTs).

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Representations for Proof-of-Possession Keys	3
3.1. Confirmation Claim	4
3.2. Representation of an Asymmetric Proof-of-Possession Key	5
3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key	6
3.4. Representation of a Key ID for a Proof-of-Possession Key	7
3.5. Specifics Intentionally Not Specified	8
4. Security Considerations	8
5. Privacy Considerations	9
6. Operational Considerations	9
7. IANA Considerations	10
7.1. CBOR Web Token Claims Registration	11
7.1.1. Registry Contents	11
7.2. CWT Confirmation Methods Registry	11
7.2.1. Registration Template	11
7.2.2. Initial Registry Contents	12
8. References	12
8.1. Normative References	12
8.2. Informative References	13
Acknowledgements	14
Document History	14
Authors' Addresses	16

1. Introduction

This specification describes how a CBOR Web Token (CWT) [RFC8392] can declare that the presenter of the CWT possesses a particular proof-of-possession (PoP) key. Proof of possession of a key is also sometimes described as being the holder-of-key. This specification

provides equivalent functionality to "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" [RFC7800] but using Concise Binary Object Representation (CBOR) [RFC7049] and CWTs [RFC8392] rather than JavaScript Object Notation (JSON) [RFC8259] and JSON Web Tokens (JWTs) [JWT].

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses terms defined in the CBOR Web Token (CWT) [RFC8392], CBOR Object Signing and Encryption (COSE) [RFC8152], and Concise Binary Object Representation (CBOR) [RFC7049] specifications.

These terms are defined by this specification:

Issuer

Party that creates the CWT and binds the claims about the subject to the proof-of-possession key.

Presenter

Party that proves possession of a private key (for asymmetric key cryptography) or secret key (for symmetric key cryptography) to a recipient of a CWT.

In the context of OAuth, this party is also called the OAuth Client.

Recipient

Party that receives the CWT containing the proof-of-possession key information from the presenter.

In the context of OAuth, this party is also called the OAuth Resource Server.

This specification provides examples in CBOR extended diagnostic notation, as defined in Appendix G of [RFC8610]. The examples include line breaks for readability.

3. Representations for Proof-of-Possession Keys

By including a "cnf" (confirmation) claim in a CWT, the issuer of the CWT declares that the presenter possesses a particular key and that the recipient can cryptographically confirm that the presenter has possession of that key. The value of the "cnf" claim is a CBOR map

(which is defined in Section 2.1 of [RFC7049]) and the members of that map identify the proof-of-possession key.

The presenter can be identified in one of several ways by the CWT, depending upon the application requirements. For instance, some applications may use the CWT "sub" (subject) claim [RFC8392], to identify the presenter. Other applications may use the "iss" (issuer) claim [RFC8392] to identify the presenter. In some applications, the subject identifier might be relative to the issuer identified by the "iss" claim. The actual mechanism used is dependent upon the application. The case in which the presenter is the subject of the CWT is analogous to Security Assertion Markup Language (SAML) 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation usage.

3.1. Confirmation Claim

The "cnf" claim in the CWT is used to carry confirmation methods. Some of them use proof-of-possession keys while others do not. This design is analogous to the SAML 2.0 [OASIS.saml-core-2.0-os] SubjectConfirmation element in which a number of different subject confirmation methods can be included (including proof-of-possession key information).

The set of confirmation members 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 confirmation members in particular ways. However, in the absence of such requirements, all confirmation members that are not understood by implementations MUST be ignored.

This specification establishes the IANA "CWT Confirmation Methods" registry for these members in Section 7.2 and registers the members defined by this specification. Other specifications can register other members used for confirmation, including other members for conveying proof-of-possession keys using different key representations.

The "cnf" claim value MUST represent only a single proof-of-possession key. At most one of the "COSE_Key" and "Encrypted_COSE_Key" confirmation values defined in Figure 1 may be present. Note that if an application needs to represent multiple proof-of-possession keys in the same CWT, one way for it to achieve this is to use other claim names, in addition to "cnf", to hold the additional proof-of-possession key information. These claims could use the same syntax and semantics as the "cnf" claim. Those claims would be defined by applications or other specifications and could be

registered in the IANA "CBOR Web Token Claims" registry [IANA.CWT.Claims].

Name	Key	Value type
COSE_Key	1	COSE_Key
Encrypted_COSE_Key	2	COSE_Encrypt or COSE_Encrypt0
kid	3	binary string

Figure 1: Summary of the cnf names, keys, and value types

3.2. Representation of an Asymmetric Proof-of-Possession Key

When the key held by the presenter is an asymmetric private key, the "COSE_Key" member is a COSE_Key [RFC8152] representing the corresponding asymmetric public key. The following example demonstrates such a declaration in the CWT Claims Set of a CWT:

```
{
  /iss/ 1 : "coaps://server.example.com",
  /aud/ 3 : "coaps://client.example.org",
  /exp/ 4 : 1879067471,
  /cnf/ 8 :{
    /COSE_Key/ 1 :{
      /kty/ 1 : /EC2/ 2,
      /crv/ -1 : /P-256/ 1,
      /x/ -2 : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa27c9
                e354089bbe13',
      /y/ -3 : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020731e
                79a3b4e47120'
    }
  }
}
```

The COSE_Key MUST contain the required key members for a COSE_Key of that key type and MAY contain other COSE_Key members, including the "kid" (Key ID) member.

The "COSE_Key" member MAY also be used for a COSE_Key representing a symmetric key, provided that the CWT is encrypted so that the key is not revealed to unintended parties. The means of encrypting a CWT is explained in [RFC8392]. If the CWT is not encrypted, the symmetric key MUST be encrypted as described in Section 3.3. This procedure is equivalent to the one defined in section 3.3 of [RFC7800].

3.3. Representation of an Encrypted Symmetric Proof-of-Possession Key

When the key held by the presenter is a symmetric key, the "Encrypted_COSE_Key" member is an encrypted COSE_Key [RFC8152] representing the symmetric key encrypted to a key known to the recipient using COSE_Encrypt or COSE_Encrypt0.

The following example illustrates a symmetric key that could subsequently be encrypted for use in the "Encrypted_COSE_Key" member:

```
{
  /kty/ 1 : /Symmetric/ 4,
  /alg/ 3 : /HMAC 256-256/ 5,
  /k/ -1 : h'6684523ab17337f173500e5728c628547cb37df
          e68449c65f885d1b73b49eae1'
}
```

The COSE_Key representation is used as the plaintext when encrypting the key.

The following example CWT Claims Set of a CWT illustrates the use of an encrypted symmetric key as the "Encrypted_COSE_Key" member value:

```
{
  /iss/ 1 : "coaps://server.example.com",
  /sub/ 2 : "24400320",
  /aud/ 3 : "s6BhdRkqt3",
  /exp/ 4 : 1311281970,
  /iat/ 5 : 1311280970,
  /cnf/ 8 : {
    /Encrypted_COSE_Key/ 2 : [
      /protected header/ h'A1010A' /{ \alg\ 1:10 \AES-CCM-16-64-128\}\/,
      /unprotected header/ { / iv / 5: h'636898994FF0EC7BFCF6D3F95B' },
      /ciphertext/ h'0573318A3573EB983E55A7C2F06CADD0796C9E584F1D0E3E
                    A8C5B052592A8B2694BE9654F0431F38D5BBC8049FA7F13F'
    ]
  }
}
```

The example above was generated with the key:

h'6162630405060708090a0b0c0d0e0f10'

3.4. Representation of a Key ID for a Proof-of-Possession Key

The proof-of-possession key can also be identified using a Key ID instead of communicating the actual key, provided the recipient is able to obtain the identified key using the Key ID. In this case, the issuer of a CWT declares that the presenter possesses a particular key and that the recipient can cryptographically confirm proof of possession of the key by the presenter by including a "cnf" claim in the CWT whose value is a CBOR map with the CBOR map containing a "kid" member identifying the key.

The following example demonstrates such a declaration in the CWT Claims Set of a CWT:

```
{
  /iss/ 1 : "coaps://as.example.com",
  /aud/ 3 : "coaps://resource.example.org",
  /exp/ 4 : 1361398824,
  /cnf/ 8 : {
    /kid/ 3 : h'dfd1aa976d8d4575a0fe34b96de2bfad'
  }
}
```

The content of the "kid" value is application specific. For instance, some applications may choose to use a cryptographic hash of the public key value as the "kid" value.

Note that the use of a Key ID to identify a proof-of-possession key needs to be carefully circumscribed, as described below and in Section 6. In cases where the Key ID is not a cryptographic value derived from the key or where not all of the parties involved are validating the cryptographic derivation, implementers should expect collisions, where different keys are assigned the same Key ID. Recipients of a CWT with a PoP key linked through only a Key ID should be prepared to handle such situations.

In the world of constrained Internet of Things (IoT) devices, there is frequently a restriction on the size of Key IDs, either because of table constraints or a desire to keep message sizes small.

Note that the value of a Key ID for a specific key is not necessarily the same for different parties. When sending a COSE encrypted message with a shared key, the Key ID may be different on both sides of the conversation, with the appropriate one being included in the message based on the recipient of the message.

3.5. Specifics Intentionally Not Specified

Proof of possession is often demonstrated by having the presenter sign a value determined by the recipient using the key possessed by the presenter. This value is sometimes called a "nonce" or a "challenge". There are, however, also other means to demonstrate freshness of the exchange and to link the proof-of-possession key to the participating parties, as demonstrated by various authentication and key exchange protocols.

The means of communicating the nonce and the nature of its contents are intentionally not described in this specification, as different protocols will communicate this information in different ways. Likewise, the means of communicating the signed nonce is also not specified, as this is also protocol specific.

Note that other means of proving possession of the key exist, which could be used in conjunction with a CWT's confirmation key. Applications making use of such alternate means are encouraged to register them in the IANA "CWT Confirmation Methods" registry established in Section 7.2.

4. Security Considerations

All the security considerations that are discussed in [RFC8392] also apply here. In addition, proof of possession introduces its own unique security issues. Possessing a key is only valuable if it is kept secret. Appropriate means must be used to ensure that unintended parties do not learn private key or symmetric key values.

Applications utilizing proof of possession SHOULD also utilize audience restriction, as described in Section 3.1.3 of [RFC8392], as it provides additional protections. Audience restriction can be used by recipients to reject messages intended for different recipients. (Of course, applications not using proof of possession can also benefit from using audience restriction to reject messages intended for different recipients.)

CBOR Web Tokens with proof-of-possession keys are used in context of an architecture, such as the ACE OAuth Framework [I-D.ietf-ace-oauth-authz], in which protocols are used by a presenter to request these tokens and to subsequently use them with recipients. Proof of possession only provides the intended security gains when the proof is known to be current and not subject to replay attacks; security protocols using mechanisms such as nonces and timestamps can be used to avoid the risk of replay when performing proof of possession for a token. Note that a discussion of the

architecture or specific protocols that CWT proof-of-possession tokens are used with is beyond the scope of this specification.

As is the case with other information included in a CWT, it is necessary to apply data origin authentication and integrity protection (via a keyed message digest or a digital signature). Data origin authentication ensures that the recipient of the CWT learns about the entity that created the CWT since this will be important for any policy decisions. Integrity protection prevents an adversary from changing any elements conveyed within the CWT payload. Special care has to be applied when carrying symmetric keys inside the CWT since those not only require integrity protection but also confidentiality protection.

As described in Section 6 (Key Identification) and Appendix D (Notes on Key Selection) of [JWS], it is important to make explicit trust decisions about the keys. Proof-of-possession signatures made with keys not meeting the application's trust criteria MUST NOT be relied upon.

5. Privacy Considerations

A proof-of-possession key can be used as a correlation handle if the same key is used on multiple occasions. Thus, for privacy reasons, it is recommended that different proof-of-possession keys be used when interacting with different parties.

6. Operational Considerations

The use of CWTs with proof-of-possession keys requires additional information to be shared between the involved parties in order to ensure correct processing. The recipient needs to be able to use credentials to verify the authenticity and integrity of the CWT. Furthermore, the recipient may need to be able to decrypt either the whole CWT or the encrypted parts thereof (see Section 3.3). This requires the recipient to know information about the issuer. Likewise, there needs to be agreement between the issuer and the recipient about the claims being used (which is also true of CWTs in general).

When an issuer creates a CWT containing a Key ID claim, it needs to make sure that it does not issue another CWT with different claims containing the same Key ID within the lifetime of the CWTs, unless intentionally desired. Failure to do so may allow one party to impersonate another party, with the potential to gain additional privileges. A case where such reuse of a Key ID would be intentional is when a presenter obtains a CWT with different claims (e.g., extended scope) for the same recipient, but wants to continue using

an existing security association (e.g., a DTLS session) bound to the key identified by the Key ID. Likewise, if PoP keys are used for multiple different kinds of CWTs in an application and the PoP keys are identified by Key IDs, care must be taken to keep the keys for the different kinds of CWTs segregated so that an attacker cannot cause the wrong PoP key to be used by using a valid Key ID for the wrong kind of CWT. Using an audience restriction for the CWT would be one strategy to mitigate this risk.

7. IANA Considerations

The following registration procedure is used for all the registries established by this specification.

Values are registered on a Specification Required [RFC8126] basis after a three-week review period on the `cwt-reg-review@ietf.org` mailing list, 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. [[Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: `cwt-reg-review@ietf.org`.]]

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to Register CWT Confirmation Method: example"). Registration requests that are undetermined for a period longer than 21 days can be brought directly to IANA's attention (using the `iana@iana.org` mailing list) for resolution.

Designated Experts should determine whether a registration request contains enough information for the registry to be populated with the new values and whether the proposed new functionality already exists. In the case of an incomplete registration or an attempt to register already existing functionality, the Designated Experts should ask for corrections or reject the registration.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification in order to enable broadly informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Experts.

7.1. CBOR Web Token Claims Registration

This specification registers the "cnf" claim in the IANA "CBOR Web Token Claims" registry [IANA.CWT.Claims] established by [RFC8392].

7.1.1. Registry Contents

- o Claim Name: "cnf"
- o Claim Description: Confirmation
- o JWT Claim Name: "cnf"
- o Claim Key: TBD (maybe 8)
- o Claim Value Type(s): map
- o Change Controller: IESG
- o Specification Document(s): Section 3.1 of [[this document]]

7.2. 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.

7.2.1. Registration Template

Confirmation Method Name:

The human-readable name requested (e.g., "kid").

Confirmation Method Description:

Brief description of the confirmation method (e.g., "Key Identifier").

JWT Confirmation Method Name:

Claim Name of the equivalent JWT confirmation method value, 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".

Confirmation Key:

CBOR map key value for the confirmation method.

Confirmation Value Type(s):

CBOR types that can be used for the confirmation method value.

Change Controller:

For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party.

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. Note that the Designated Experts and IANA must be able to obtain copies of the specification document(s) to perform their work.

7.2.2. Initial Registry Contents

- o Confirmation Method Name: "COSE_Key"
- o Confirmation Method Description: COSE_Key Representing Public Key
- o JWT Confirmation Method Name: "jwk"
- o Confirmation Key: 1
- o Confirmation Value Type(s): COSE_Key structure
- o Change Controller: IESG
- o Specification Document(s): Section 3.2 of [[this document]]

- o Confirmation Method Name: "Encrypted_COSE_Key"
- o Confirmation Method Description: Encrypted COSE_Key
- o JWT Confirmation Method Name: "jwe"
- o Confirmation Key: 2
- o Confirmation Value Type(s): COSE_Encrypt or COSE_Encrypt0 structure (with an optional corresponding COSE_Encrypt or COSE_Encrypt0 tag)
- o Change Controller: IESG
- o Specification Document(s): Section 3.3 of [[this document]]

- o Confirmation Method Name: "kid"
- o Confirmation Method Description: Key Identifier
- o JWT Confirmation Method Name: "kid"
- o Confirmation Key: 3
- o Confirmation Value Type(s): binary string
- o Change Controller: IESG
- o Specification Document(s): Section 3.4 of [[this document]]

8. References**8.1. Normative References****[IANA.CWT.Claims]**

IANA, "CBOR Web Token Claims",
<<http://www.iana.org/assignments/cwt>>.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

8.2. Informative 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) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-21 (work in progress), February 2019.
- [IANA.JWT.Claims]
IANA, "JSON Web Token Claims", <<http://www.iana.org/assignments/jwt>>.
- [JWS] 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>>.
- [JWT] 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>>.
- [OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005, <<http://docs.oasis-open.org/security/saml/v2.0/>>.

- [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, <<https://www.rfc-editor.org/info/rfc7800>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

Acknowledgements

Thanks to the following people for their reviews of the specification: Roman Danyliw, Christer Holmberg, Benjamin Kaduk, Mirja Kuehlewind, Yoav Nir, Michael Richardson, Adam Roach, Eric Vyncke, and Jim Schaad.

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

Document History

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

-11

- o Addressed remaining IESG review comment by Mirja Kuehlewind.

-10

- o Addressed IESG review comments by Adam Roach and Eric Vyncke.

-09

- o Addressed Gen-ART review comments by Christer Holmberg and SecDir review comments by Yoav Nir.

-08

- o Addressed remaining Area Director review comments by Benjamin Kaduk.

-07

- o Addressed Area Director review by Benjamin Kaduk.

-06

- o Corrected nits identified by Roman Danyliw.

-05

- o Added text suggested by Jim Schaad describing considerations when using the Key ID confirmation method.

-04

- o Addressed additional WGLC comments by Jim Schaad and Roman Danyliw.

-03

- o Addressed review comments by Jim Schaad, see <https://www.ietf.org/mail-archive/web/ace/current/msg02798.html>
- o Removed unnecessary sentence in the introduction regarding the use any strings that could be case-sensitive.
- o Clarified the terms Presenter and Recipient.
- o Clarified text about the confirmation claim.

-02

- o Changed "typically" to "often" when describing ways of performing proof of possession.
- o Changed b64 to hex encoding in an example.
- o Changed to using the RFC 8174 boilerplate instead of the RFC 2119 boilerplate.

-01

- o Now uses CBOR diagnostic notation for the examples.
- o Added a table summarizing the "cnf" names, keys, and value types.
- o Addressed some of Jim Schaad's feedback on -00.

-00

- o Created the initial working group draft from draft-jones-ace-cwt-proof-of-possession-01.

Authors' Addresses

Michael B. Jones
Microsoft

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

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@ri.se

Goeran Selander
Ericsson AB
Faeroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Samuel Erdtman
Spotify

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: 6 December 2021

S. Gerdes
O. Bergmann
C. Bormann
Universität Bremen TZI
G. Selander
Ericsson AB
L. Seitz
Combitech
4 June 2021

Datagram Transport Layer Security (DTLS) Profile for Authentication and
Authorization for Constrained Environments (ACE)
draft-ietf-ace-dtls-authorize-18

Abstract

This specification defines a profile of the ACE framework that allows constrained servers to delegate client authentication and authorization. The protocol relies on DTLS version 1.2 for communication security between entities in a constrained network using either raw public keys or pre-shared keys. A resource-constrained server 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 <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on 6 December 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Terminology	4
2. Protocol Overview	4
3. Protocol Flow	6
3.1. Communication Between the Client and the Authorization Server	6
3.2. Raw Public Key Mode	7
3.2.1. Access Token Retrieval from the Authorization Server	7
3.2.2. DTLS Channel Setup Between Client and Resource Server	9
3.3. PreSharedKey Mode	10
3.3.1. Access Token Retrieval from the Authorization Server	11
3.3.2. DTLS Channel Setup Between Client and Resource Server	15
3.4. Resource Access	17
4. Dynamic Update of Authorization Information	19
5. Token Expiration	20
6. Secure Communication with an Authorization Server	20
7. Security Considerations	21
7.1. Reuse of Existing Sessions	23
7.2. Multiple Access Tokens	23
7.3. Out-of-Band Configuration	23
8. Privacy Considerations	24
9. IANA Considerations	24
10. Acknowledgments	25
11. References	25
11.1. Normative References	25
11.2. Informative References	27
Authors' Addresses	28

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 version 1.2 [RFC6347] to communicate. This specification uses DTLS 1.2 terminology, but later versions such as DTLS 1.3 can be used instead. The client obtains an access token, bound to a key (the proof-of-possession key), from an authorization server to prove its authorization to access protected resources hosted by the resource server. Also, the client and the resource server are provided by the authorization server with the necessary keying material to establish a DTLS session. The communication between client and authorization server may also be secured with DTLS. This specification supports DTLS with Raw Public Keys (RPK) [RFC7250] and with Pre-Shared Keys (PSK) [RFC4279]. How token introspection [RFC7662] is performed between RS and AS is out of scope for this specification.

The ACE framework requires that client and server mutually authenticate each other before any application data is exchanged. DTLS enables mutual authentication if both client and server prove their ability to use certain keying material in the DTLS handshake. The authorization server assists in this process on the server side by incorporating keying material (or information about keying material) into the access token, which is considered a "proof of possession" token.

In the RPK mode, the client proves that it can use the RPK bound to the token and the server shows that it can use a certain RPK.

The resource server needs access to the token in order to complete this exchange. For the RPK mode, the client must upload the access token to the resource server before initiating the handshake, as described in Section 5.10.1 of the ACE framework [I-D.ietf-ace-oauth-authz].

In the PSK mode, client and server show with the DTLS handshake that they can use the keying material that is bound to the access token. To transfer the access token from the client to the resource server, the "psk_identity" parameter in the DTLS PSK handshake may be used instead of uploading the token prior to the handshake.

As recommended in Section 5.8 of [I-D.ietf-ace-oauth-authz], this specification uses CBOR web tokens to convey claims within an access token issued by the server. While other formats could be used as well, those are out of scope for this document.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

The authorization information (authz-info) resource refers to the authorization information endpoint as specified in [I-D.ietf-ace-oauth-authz]. The term "claim" is used in this document with the same semantics as in [I-D.ietf-ace-oauth-authz], i.e., it denotes information carried in the access token or returned from introspection.

2. Protocol Overview

The CoAP-DTLS profile for ACE specifies the transfer of authentication information and, if necessary, authorization information between the client (C) and the resource server (RS) during setup of a DTLS session for CoAP messaging. It also specifies how the client can use CoAP over DTLS to retrieve an access token from the authorization server (AS) for a protected resource hosted on the resource server. As specified in Section 6.7 of [I-D.ietf-ace-oauth-authz], use of DTLS for one or both of these interactions is completely independent.

This profile requires the client to retrieve an access token for protected resource(s) it wants to access on the resource server 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):

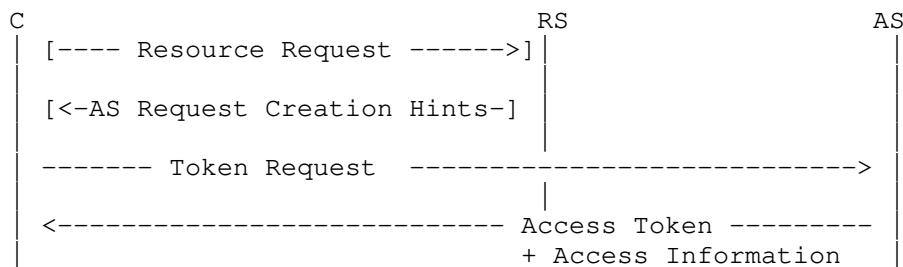


Figure 1: Retrieving an Access Token

To determine the authorization server in charge of a resource hosted at the resource server, the client can send an initial Unauthorized Resource Request message to the resource server. The resource server then denies the request and sends an AS Request Creation Hints message containing the address of its authorization server back to the client as specified in Section 5.3 of [I-D.ietf-ace-oauth-authz].

Once the client knows the authorization server's address, it can send an access token request to the token endpoint at the authorization server as specified in [I-D.ietf-ace-oauth-authz]. As the access token request as well as the response may contain confidential data, the communication between the client and the authorization server must be confidentiality-protected and ensure authenticity. The client is expected to have been registered at the authorization server as outlined in Section 4 of [I-D.ietf-ace-oauth-authz].

The access token returned by the authorization server can then be used by the client to establish a new DTLS session with the resource server. When the client intends to use an asymmetric proof-of-possession key in the DTLS handshake with the resource server, the client **MUST** upload the access token to the authz-info resource, i.e. the authz-info endpoint, on the resource server before starting the DTLS handshake, as described in Section 5.10.1 of [I-D.ietf-ace-oauth-authz]. In case the client uses a symmetric proof-of-possession key in the DTLS handshake, the procedure as above **MAY** be used, or alternatively, the access token **MAY** instead be transferred in the DTLS ClientKeyExchange message (see Section 3.3.2). In any case, DTLS **MUST** be used in a mode that provides replay protection.

Figure 2 depicts the common protocol flow for the DTLS profile after the client has retrieved the access token from the authorization server, AS.

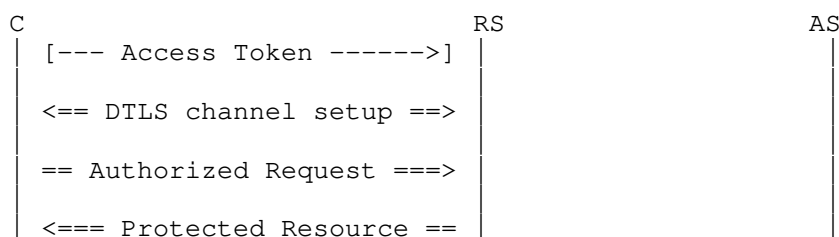


Figure 2: Protocol overview

3. Protocol Flow

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

Section 3.1 describes how the communication between the client (C) and the authorization server (AS) must be secured. Depending on the used CoAP security mode (see also Section 9 of [RFC7252], the Client-to-AS request, AS-to-Client response and DTLS session establishment carry slightly different information. Section 3.2 addresses the use of raw public keys while Section 3.3 defines how pre-shared keys are used in this profile.

3.1. Communication Between the Client and the Authorization Server

To retrieve an access token for the resource that the client wants to access, the client requests an access token from the authorization server. Before the client can request the access token, the client and the authorization server MUST establish a secure communication channel. This profile assumes that the keying material to secure this communication channel has securely been obtained either by manual configuration or in an automated provisioning process. The following requirements in alignment with Section 6.5 of [I-D.ietf-ace-oauth-authz] therefore must be met:

- * The client MUST securely have obtained keying material to communicate with the authorization server.
- * Furthermore, the client MUST verify that the authorization server is authorized to provide access tokens (including authorization information) about the resource server to the client, and that this authorization information about the authorization server is still valid.
- * Also, the authorization server MUST securely have obtained keying material for the client, and obtained authorization rules approved by the resource owner (RO) concerning the client and the resource server that relate to this keying material.

The client and the authorization server MUST use their respective keying material for all exchanged messages. How the security association between the client and the authorization server is bootstrapped is not part of this document. The client and the authorization server must ensure the confidentiality, integrity and authenticity of all exchanged messages within the ACE protocol.

Section 6 specifies how communication with the authorization server is secured.

3.2. Raw Public Key Mode

When the client uses raw public key authentication, the procedure is as described in the following.

3.2.1. Access Token Retrieval from the Authorization Server

After the client and the authorization server mutually authenticated each other and validated each other's authorization, the client sends a token request to the authorization server's token endpoint. The client MUST add a "req_cnf" object carrying either its raw public key or a unique identifier for a public key that it has previously made known to the authorization server. It is RECOMMENDED that the client uses DTLS with the same keying material to secure the communication with the authorization server, proving possession of the key as part of the token request. Other mechanisms for proving possession of the key may be defined in the future.

An example access token request from the client to the authorization server is depicted in Figure 3.

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  grant_type : client_credentials,
  audience   : "tempSensor4711",
  req_cnf    : {
    COSE_Key : {
      kty : EC2,
      crv : P-256,
      x   : h'e866c35f4c3c81bb96a1...',
      y   : h'2e25556be097c8778a20...'
    }
  }
}
```

Figure 3: Access Token Request Example for RPK Mode

The example shows an access token request for the resource identified by the string "tempSensor4711" on the authorization server using a raw public key.

The authorization server MUST check if the client that it communicates with is associated with the RPK in the "req_cnf" parameter before issuing an access token to it. If the authorization server determines that the request is to be authorized according to the respective authorization rules, it generates an access token response for the client. The access token MUST be bound to the RPK of the client by means of the "cnf" claim.

The response MUST contain an "ace_profile" parameter if the "ace_profile" parameter in the request is empty, and MAY contain this parameter otherwise (see Section 5.8.2 of [I-D.ietf-ace-oauth-authz]). This parameter is set to "coap_dtls" to indicate that this profile MUST be used for communication between the client and the resource server. The response also contains an access token with information for the resource server about the client's public key. The authorization server MUST return in its response the parameter "rs_cnf" unless it is certain that the client already knows the public key of the resource server. The authorization server MUST ascertain that the RPK specified in "rs_cnf" belongs to the resource server that the client wants to communicate with. The authorization server MUST protect the integrity of the access token such that the resource server can detect unauthorized changes. If the access token contains confidential data, the authorization server MUST also protect the confidentiality of the access token.

The client MUST ascertain that the access token response belongs to a certain previously sent access token request, as the request may specify the resource server with which the client wants to communicate.

An example access token response from the authorization server to the client is depicted in Figure 4. Here, the contents of the "access_token" claim have been truncated to improve readability. The response comprises access information for the client that contains the server's public key in the "rs_cnf" parameter. Caching proxies process the Max-Age option in the CoAP response which has a default value of 60 seconds (Section 5.6.1 of [RFC7252]). The authorization server SHOULD adjust the Max-Age option such that it does not exceed the "expires_in" parameter to avoid stale responses.

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 3560
Payload:
{
  access_token : b64'SlAV32hkKG...
    (remainder of CWT omitted for brevity;
    CWT contains the client's RPK in the cnf claim)',
  expires_in : 3600,
  rs_cnf      : {
    COSE_Key : {
      kty : EC2,
      crv : P-256,
      x   : h'd7cc072de2205bdc1537...',
      y   : h'f95e1d4b851a2cc80fff...'
    }
  }
}
```

Figure 4: Access Token Response Example for RPK Mode

3.2.2. DTLS Channel Setup Between Client and Resource Server

Before the client initiates the DTLS handshake with the resource server, the client MUST send a "POST" request containing the obtained access token to the authz-info resource hosted by the resource server. After the client receives a confirmation that the resource server has accepted the access token, it proceeds to establish a new DTLS channel with the resource server. The client MUST use its correct public key in the DTLS handshake. If the authorization server has specified a "cnf" field in the access token response, the client MUST use this key. Otherwise, the client MUST use the public key that it specified in the "req_cnf" of the access token request. The client MUST specify this public key in the SubjectPublicKeyInfo structure of the DTLS handshake as described in [RFC7250].

If the client does not have the keying material belonging to the public key, the client MAY try to send an access token request to the AS where it specifies its public key in the "req_cnf" parameter. If the AS still specifies a public key in the response that the client does not have, the client SHOULD re-register with the authorization server to establish a new client public key. This process is out of scope for this document.

To be consistent with [RFC7252], which allows for shortened MAC tags in constrained environments, an implementation that supports the RPK mode of this profile MUST at least support the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. As discussed in

[RFC7748], new ECC curves have been defined recently that are considered superior to the so-called NIST curves. Implementations of this profile therefore MUST implement support for curve25519 (cf. [RFC8032], [RFC8422]) as this curve said to be efficient and less dangerous regarding implementation errors than the secp256r1 curve mandated in [RFC7252].

The resource server MUST check if the access token is still valid, if the resource server is the intended destination (i.e., the audience) of the token, and if the token was issued by an authorized authorization server (see also section 5.10.1.1 of [I-D.ietf-ace-oauth-authz]). The access token is constructed by the authorization server such that the resource server can associate the access token with the Client's public key. The "cnf" claim MUST contain either the client's RPK or, if the key is already known by the resource server (e.g., from previous communication), a reference to this key. If the authorization server has no certain knowledge that the Client's key is already known to the resource server, the Client's public key MUST be included in the access token's "cnf" parameter. If CBOR web tokens [RFC8392] are used (as recommended in [I-D.ietf-ace-oauth-authz]), keys MUST be encoded as specified in [RFC8747]. A resource server MUST have the capacity to store one access token for every proof-of-possession key of every authorized client.

The raw public key used in the DTLS handshake with the client MUST belong to the resource server. If the resource server has several raw public keys, it needs to determine which key to use. The authorization server can help with this decision by including a "cnf" parameter in the access token that is associated with this communication. In this case, the resource server MUST use the information from the "cnf" field to select the proper keying material.

Thus, the handshake only finishes if the client and the resource server are able to use their respective keying material.

3.3. PreSharedKey Mode

When the client uses pre-shared key authentication, the procedure is as described in the following.

3.3.1. Access Token Retrieval from the Authorization Server

To retrieve an access token for the resource that the client wants to access, the client MAY include a "cnf" object carrying an identifier for a symmetric key in its access token request to the authorization server. This identifier can be used by the authorization server to determine the shared secret to construct the proof-of-possession token. The authorization server MUST check if the identifier refers to a symmetric key that was previously generated by the authorization server as a shared secret for the communication between this client and the resource server. If no such symmetric key was found, the authorization server MUST generate a new symmetric key that is returned in its response to the client.

The authorization server MUST determine the authorization rules for the client it communicates with as defined by the resource owner and generate the access token accordingly. If the authorization server authorizes the client, it returns an AS-to-Client response. If the "ace_profile" parameter is present, it is set to "coap_dtls". The authorization server MUST ascertain that the access token is generated for the resource server that the client wants to communicate with. Also, the authorization server MUST protect the integrity of the access token to ensure that the resource server can detect unauthorized changes. If the token contains confidential data such as the symmetric key, the confidentiality of the token MUST also be protected. Depending on the requested token type and algorithm in the access token request, the authorization server adds access information to the response that provides the client with sufficient information to setup a DTLS channel with the resource server. The authorization server adds a "cnf" parameter to the access information carrying a "COSE_Key" object that informs the client about the shared secret that is to be used between the client and the resource server. To convey the same secret to the resource server, the authorization server can include it directly in the access token by means of the "cnf" claim or provide sufficient information to enable the resource server to derive the shared secret from the access token. As an alternative, the resource server MAY use token introspection to retrieve the keying material for this access token directly from the authorization server.

An example access token request for an access token with a symmetric proof-of-possession key is illustrated in Figure 5.

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  audience      : "smokeSensor1807",
}
```

Figure 5: Example Access Token Request, (implicit) symmetric PoP-key

A corresponding example access token response is illustrated in Figure 6. In this example, the authorization server returns a 2.01 response containing a new access token (truncated to improve readability) and information for the client, including the symmetric key in the cnf claim. The information is transferred as a CBOR data structure as specified in [I-D.ietf-ace-oauth-authz].

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 85800
Payload:
{
  access_token : h'd08343a10...
  (remainder of CWT omitted for brevity)
  token_type   : PoP,
  expires_in   : 86400,
  profile      : coap_dtls,
  cnf          : {
    COSE_Key : {
      kty : symmetric,
      kid : h'3d027833fc6267ce',
      k   : h'73657373696f6e6b6579'
    }
  }
}
```

Figure 6: Example Access Token Response, symmetric PoP-key

The access token also comprises a "cnf" claim. This claim usually contains a "COSE_Key" object [RFC8152] that carries either the symmetric key itself or a key identifier that can be used by the resource server to determine the secret key it shares with the client. If the access token carries a symmetric key, the access token MUST be encrypted using a "COSE_Encrypt0" structure (see section 7.1 of [RFC8392]). The authorization server MUST use the keying material shared with the resource server to encrypt the token.

The "cnf" structure in the access token is provided in Figure 7.

```
cnf : {  
  COSE_Key : {  
    kty : symmetric,  
    kid : h'3d027833fc6267ce'  
  }  
}
```

Figure 7: Access Token without Keying Material

A response that declines any operation on the requested resource is constructed according to Section 5.2 of [RFC6749], (cf. Section 5.8.3. of [I-D.ietf-ace-oauth-authz]). Figure 8 shows an example for a request that has been rejected due to invalid request parameters.

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

Figure 8: Example Access Token Response With Reject

The method for how the resource server determines the symmetric key from an access token containing only a key identifier is application-specific; the remainder of this section provides one example.

The authorization server and the resource server are assumed to share a key derivation key used to derive the symmetric key shared with the client from the key identifier in the access token. The key derivation key may be derived from some other secret key shared between the authorization server and the resource server. This key needs to be securely stored and processed in the same way as the key used to protect the communication between the authorization server and the resource server.

Knowledge of the symmetric key shared with the client must not reveal any information about the key derivation key or other secret keys shared between the authorization server and resource server.

In order to generate a new symmetric key to be used by client and resource server, the authorization server generates a new key identifier which **MUST** be unique among all key identifiers used by the authorization server for this resource server. The authorization server then uses the key derivation key shared with the resource server to derive the symmetric key as specified below. Instead of providing the keying material in the access token, the authorization

server includes the key identifier in the "kid" parameter, see Figure 7. This key identifier enables the resource server to calculate the symmetric key used for the communication with the client using the key derivation key and a KDF to be defined by the application, for example HKDF-SHA-256. The key identifier picked by the authorization server MUST be unique for each access token where a unique symmetric key is required.

In this example, HKDF consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]. The symmetric key is derived from the key identifier, the key derivation key and other data:

```
OKM = HKDF(salt, IKM, info, L),
```

where:

- * OKM, the output keying material, is the derived symmetric key
- * salt is the empty byte string
- * IKM, the input keying material, is the key derivation key as defined above
- * info is the serialization of a CBOR array consisting of ([RFC8610]):

```
info = [  
    type : tstr,  
    L : uint,  
    access_token: bytes  
]
```

where:

- * type is set to the constant text string "ACE-CoAP-DTLS-key-derivation",
- * L is the size of the symmetric key in bytes,
- * access_token is the content of the "access_token" field as transferred from the authorization server to the resource server.

All CBOR data types are encoded in CBOR using preferred serialization and deterministic encoding as specified in Section 4 of [RFC8949]. This implies in particular that the "type" and "L" components use the minimum length encoding. The content of the "access_token" field is treated as opaque data for the purpose of key derivation.

Use of a unique (per resource server) "kid" and the use of a key derivation IKM that MUST be unique per authorization server/resource server pair as specified above will ensure that the derived key is not shared across multiple clients. However, to provide variation in the derived key across different tokens used by the same client, it is additionally RECOMMENDED to include the "iat" claim and either the "exp" or "exp_iat" claims in the access token.

3.3.2. DTLS Channel Setup Between Client and Resource Server

When a client receives an access token response from an authorization server, the client MUST check if the access token response is bound to a certain previously sent access token request, as the request may specify the resource server with which the client wants to communicate.

The client checks if the payload of the access token response contains an "access_token" parameter and a "cnf" parameter. With this information the client can initiate the establishment of a new DTLS channel with a resource server. To use DTLS with pre-shared keys, the client 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. To be consistent with the recommendations in [RFC7252], a client in the PSK mode MUST support the cipher suite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655].

In PreSharedKey mode, the knowledge of the shared secret by the client and the resource server is used for mutual authentication between both peers. Therefore, the resource server must be able to determine the shared secret from the access token. Following the general ACE authorization framework, the client can upload the access token to the resource server's authz-info resource before starting the DTLS handshake. The client then needs to indicate during the DTLS handshake which previously uploaded access token it intends to use. To do so, it MUST create a "COSE_Key" structure with the "kid" that was conveyed in the "rs_cnf" claim in the token response from the authorization server and the key type "symmetric". This structure then is included as the only element in the "cnf" structure whose CBOR serialization is used as value for "psk_identity" as shown in Figure 9.


```
{ cnf : {  
  COSE_Key : {  
    kty: symmetric,  
    kid : h'3d027833fc6267ce'  
  }  
}
```

Figure 9: Access token containing a single kid parameter

The actual CBOR serialization for the data structure from Figure 9 as sequence of bytes in hexadecimal notation will be:

A1 08 A1 01 A2 01 04 02 48 3D 02 78 33 FC 62 67 CE

As an alternative to the access token upload, the client can provide the most recent access token in the "psk_identity" field of the ClientKeyExchange message. To do so, the client MUST treat the contents of the "access_token" field from the AS-to-Client response as opaque data as specified in Section 4.2 of [RFC7925] and not perform any re-coding. This allows the resource server to retrieve the shared secret directly from the "cnf" claim of the access token.

If a resource server receives a ClientKeyExchange message that contains a "psk_identity" with a length greater than zero, it MUST parse the contents of the "psk_identity" field as CBOR data structure and process the contents as following:

- * If the data contains a "cnf" field with a "COSE_Key" structure with a "kid", the resource server continues the DTLS handshake with the associated key that corresponds to this kid.
- * If the data comprises additional CWT information, this information must be stored as an access token for this DTLS association before continuing with the DTLS handshake.

If the contents of the "psk_identity" do not yield sufficient information to select a valid access token for the requesting client, the resource server aborts the DTLS handshake with an "illegal_parameter" alert.

When the resource server receives an access token, it MUST check if the access token is still valid, if the resource server is the intended destination (i.e., the audience of the token), and if the token was issued by an authorized authorization server. This specification implements access tokens as proof-of-possession tokens. Therefore, the access token is bound to a symmetric PoP key that is used as shared secret between the client and the resource server. A

resource server MUST have the capacity to store one access token for every proof-of-possession key of every authorized client. The resource server may use token introspection [RFC7662] on the access token to retrieve more information about the specific token. The use of introspection is out of scope for this specification.

While the client can retrieve the shared secret from the contents of the "cnf" parameter in the AS-to-Client response, the resource server uses the information contained in the "cnf" claim of the access token to determine the actual secret when no explicit "kid" was provided in the "psk_identity" field. If key derivation is used, the "cnf" claim MUST contain a "kid" parameter to be used by the server as the IKM for key derivation as described above.

3.4. Resource Access

Once a DTLS channel has been established as described in Section 3.2 or Section 3.3, respectively, the client is authorized to access resources covered by the access token it has uploaded to the authz-info resource hosted by the resource server.

With the successful establishment of the DTLS channel, the client and the resource server have proven that they can use their respective keying material. An access token that is bound to the client's keying material is associated with the channel. According to Section 5.10.1 of [I-D.ietf-ace-oauth-authz], there should be only one access token for each client. New access tokens issued by the authorization server SHOULD replace previously issued access tokens for the respective client. The resource server therefore needs a common understanding with the authorization server how access tokens are ordered. The authorization server may, e.g., specify a "cti" claim for the access token (see Section 5.9.4 of [I-D.ietf-ace-oauth-authz]) to employ a strict order.

Any request that the resource server receives on a DTLS channel that is tied to an access token via its keying material MUST be checked against the authorization rules that can be determined with the access token. The resource server MUST check for every request if the access token is still valid. If the token has expired, the resource server MUST remove it. Incoming CoAP requests that are not authorized with respect to any access token that is associated with the client MUST be rejected by the resource server with 4.01 response. The response SHOULD include AS Request Creation Hints as described in Section 5.2 of [I-D.ietf-ace-oauth-authz].

The resource server MUST NOT accept an incoming CoAP request as authorized if any of the following fails:

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 client is valid.
3. The request is destined for the resource server.
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 that are not thus authorized **MUST** be rejected according to Section 5.10.1.1 of [I-D.ietf-ace-oauth-authz]

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.

The client **MUST** ascertain that its keying material is still valid before sending a request or processing a response. If the client recently has updated the access token (see Section 4), it must be prepared that its request is still handled according to the previous authorization rules as there is no strict ordering between access token uploads and resource access messages. See also Section 7.2 for a discussion of access token processing.

If the client gets an error response containing AS Request Creation Hints (cf. Section 5.3 of [I-D.ietf-ace-oauth-authz]) as response to its requests, it **SHOULD** request a new access token from the authorization server in order to continue communication with the resource server.

Unauthorized requests that have been received over a DTLS session **SHOULD** be treated as non-fatal by the resource server, i.e., the DTLS session **SHOULD** be kept alive until the associated access token has expired.

4. Dynamic Update of Authorization Information

Resource servers must only use a new access token to update the authorization information for a DTLS session if the keying material that is bound to the token is the same that was used in the DTLS handshake. By associating the access tokens with the identifier of an existing DTLS session, the authorization information can be updated without changing the cryptographic keys for the DTLS communication between the client and the resource server, i.e. an existing session can be used with updated permissions.

The client can therefore update the authorization information stored at the resource server at any time without changing an established DTLS session. To do so, the client requests a new access token from the authorization server for the intended action on the respective resource and uploads this access token to the authz-info resource on the resource server.

Figure 10 depicts the message flow where the client requests a new access token after a security association between the client and the resource server has been established using this protocol. If the client wants to update the authorization information, the token request MUST specify the key identifier of the proof-of-possession key used for the existing DTLS channel between the client and the resource server in the "kid" parameter of the Client-to-AS request. The authorization server MUST verify that the specified "kid" denotes a valid verifier for a proof-of-possession token that has previously been issued to the requesting client. Otherwise, the Client-to-AS request MUST be declined with the error code "unsupported_pop_key" as defined in Section 5.8.3 of [I-D.ietf-ace-oauth-authz].

When the authorization server issues a new access token to update existing authorization information, it MUST include the specified "kid" parameter in this access token. A resource server MUST replace the authorization information of any existing DTLS session that is identified by this key identifier with the updated authorization information.

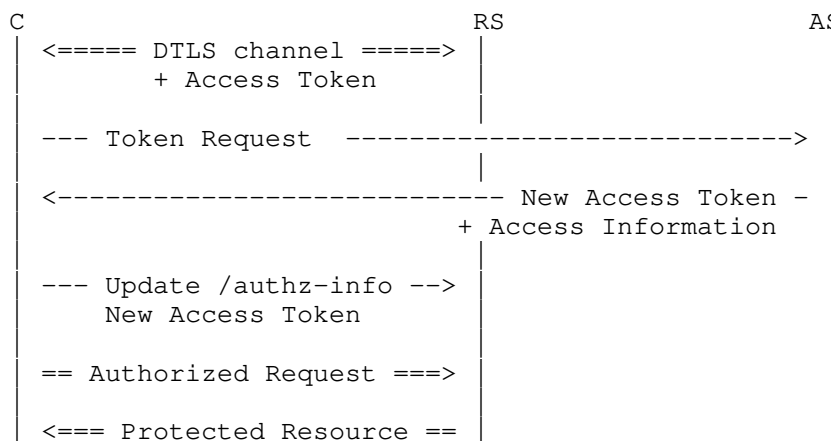


Figure 10: Overview of Dynamic Update Operation

5. Token Expiration

The resource server MUST delete access tokens that are no longer valid. DTLS associations that have been setup in accordance with this profile are always tied to specific tokens (which may be exchanged with a dynamic update as described in Section 4). As tokens may become invalid at any time (e.g., because they have expired), the association may become useless at some point. A resource server therefore MUST terminate existing DTLS association after the last access token associated with this association has expired.

As specified in Section 5.10.3 of [I-D.ietf-ace-oauth-authz], the resource server MUST notify the client with an error response with code 4.01 (Unauthorized) for any long running request before terminating the association.

6. Secure Communication with an Authorization Server

As specified in the ACE framework (Sections 5.8 and 5.9 of [I-D.ietf-ace-oauth-authz]), the requesting entity (the resource server and/or the client) and the authorization server communicate via the token endpoint or introspection endpoint. The use of CoAP and DTLS for this communication is RECOMMENDED in this profile. Other protocols fulfilling the security requirements defined in Section 5 of [I-D.ietf-ace-oauth-authz] MAY be used instead.

How credentials (e.g., PSK, RPK, X.509 cert) for using DTLS with the authorization server are established is out of scope for this profile.

If other means of securing the communication with the authorization server are used, the communication security requirements from Section 6.2 of [I-D.ietf-ace-oauth-authz] remain applicable.

7. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. As it follows this framework's general approach, the general security considerations from Section 6 of [I-D.ietf-ace-oauth-authz] also apply to this profile.

The authorization server must ascertain that the keying material for the client that it provides to the resource server actually is associated with this client. Malicious clients may hand over access tokens containing their own access permissions to other entities. This problem cannot be completely eliminated. Nevertheless, in RPK mode it should not be possible for clients to request access tokens for arbitrary public keys: if the client can cause the authorization server to issue a token for a public key without proving possession of the corresponding private key, this allows for identity misbinding attacks where the issued token is usable by an entity other than the intended one. The authorization server therefore at some point needs to validate that the client can actually use the private key corresponding to the client's public key.

When using pre-shared keys provisioned by the authorization server, the security level depends on the randomness of PSK, and the security of the TLS cipher suite and key exchange algorithm. As this specification targets at constrained environments, message payloads exchanged between the client and the resource server are expected to be small and rare. CoAP [RFC7252] mandates the implementation of cipher suites with abbreviated, 8-byte tags for message integrity protection. For consistency, this profile requires implementation of the same cipher suites. For application scenarios where the cost of full-width authentication tags is low compared to the overall amount of data being transmitted, the use of cipher suites with 16-byte integrity protection tags is preferred.

The PSK mode of this profile offers a distribution mechanism to convey authorization tokens together with a shared secret to a client and a server. As this specification aims at constrained devices and uses CoAP [RFC7252] as transfer protocol, at least the cipher suite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655] should be supported. The access tokens and the corresponding shared secrets generated by the authorization server are expected to be sufficiently short-lived to provide similar forward-secrecy properties to using ephemeral Diffie-Hellman (DHE) key exchange mechanisms. For longer-lived access tokens, DHE cipher suites should be used, i.e., cipher suites of the form TLS_DHE_PSK_*.

Constrained devices that use DTLS [RFC6347] are inherently vulnerable to Denial of Service (DoS) attacks as the handshake protocol requires creation of internal state within the device. This is specifically of concern where an adversary is able to intercept the initial cookie exchange and interject forged messages with a valid cookie to continue with the handshake. A similar issue exists with the unprotected authorization information endpoint when the resource server needs to keep valid access tokens for a long time. Adversaries could fill up the constrained resource server's internal storage for a very long time with interjected or otherwise retrieved valid access tokens. To mitigate against this, the resource server should set a time boundary until an access token that has not been used until then will be deleted.

The protection of access tokens that are stored in the authorization information endpoint depends on the keying material that is used between the authorization server and the resource server: The resource server must ensure that it processes only access tokens that are (encrypted and) integrity-protected by an authorization server that is authorized to provide access tokens for the resource server.

7.1. Reuse of Existing Sessions

To avoid the overhead of a repeated DTLS handshake, [RFC7925] recommends session resumption [RFC8446] to reuse session state from an earlier DTLS association and thus requires client side implementation. In this specification, the DTLS session is subject to the authorization rules denoted by the access token that was used for the initial setup of the DTLS association. Enabling session resumption would require the server to transfer the authorization information with the session state in an encrypted SessionTicket to the client. Assuming that the server uses long-lived keying material, this could open up attacks due to the lack of forward secrecy. Moreover, using this mechanism, a client can resume a DTLS session without proving the possession of the PoP key again. Therefore, session resumption should be used only in combination with reasonably short-lived PoP keys.

Since renegotiation of DTLS associations is prone to attacks as well, [RFC7925] requires clients to decline any renegotiation attempt. A server that wants to initiate re-keying therefore SHOULD periodically force a full handshake.

7.2. Multiple Access Tokens

Developers SHOULD avoid using multiple access tokens for a client (see also section 5.10.1 of [I-D.ietf-ace-oauth-authz]).

Even when a single access token per client is used, an attacker could compromise the dynamic update mechanism for existing DTLS connections by delaying or reordering packets destined for the authz-info endpoint. Thus, the order in which operations occur at the resource server (and thus which authorization info is used to process a given client request) cannot be guaranteed. Especially in the presence of later-issued access tokens that reduce the client's permissions from the initial access token, it is impossible to guarantee that the reduction in authorization will take effect prior to the expiration of the original token.

7.3. Out-of-Band Configuration

To communicate securely, the authorization server, the client and the resource server require certain information that must be exchanged outside the protocol flow described in this document. The authorization server must have obtained authorization information concerning the client and the resource server that is approved by the resource owner as well as corresponding keying material. The resource server must have received authorization information approved by the resource owner concerning its authorization managers and the

respective keying material. The client must have obtained authorization information concerning the authorization server approved by its owner as well as the corresponding keying material. Also, the client's owner must have approved of the client's communication with the resource server. The client and the authorization server must have obtained a common understanding how this resource server is identified to ensure that the client obtains access token and keying material for the correct resource server. If the client is provided with a raw public key for the resource server, it must be ascertained to which resource server (which identifier and authorization information) the key is associated. All authorization information and keying material must be kept up to date.

8. Privacy Considerations

This privacy considerations from Section 7 of the [I-D.ietf-ace-oauth-authz] apply also to this profile.

An unprotected response to an unauthorized request may disclose information about the resource server and/or its existing relationship with the client. It is advisable to include as little information as possible in an unencrypted response. When a DTLS session between an authenticated client and the resource server already exists, more detailed information MAY be included with an error response to provide the client with sufficient information to react on that particular error.

Also, unprotected requests to the resource server may reveal information about the client, e.g., which resources the client attempts to request or the data that the client wants to provide to the resource server. The client SHOULD NOT send confidential data in an unprotected request.

Note that some information might still leak after DTLS session is established, due to observable message sizes, the source, and the destination addresses.

9. IANA Considerations

The following registrations are done for the ACE OAuth Profile Registry following the procedure specified in [I-D.ietf-ace-oauth-authz].

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

Profile name: coap_dtls

Profile Description: Profile for delegating client authentication and authorization in a constrained environment by establishing a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes.

Profile ID: TBD (suggested: 1)

Change Controller: IESG

Reference: [RFC-XXXX]

10. Acknowledgments

Special thanks to Jim Schaad for his contributions and reviews of this document and to Ben Kaduk for his thorough reviews of this document. Thanks also to Paul Kyzivat for his review. The authors also would like to thank Marco Tiloca for his contributions.

Ludwig Seitz worked on this document as part of the CelticNext projects CyberWI, and CRITISEC with funding from Vinnova.

11. References

11.1. 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) using the OAuth 2.0 Framework (ACE-OAuth)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-41, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-41.txt>>.

[I-D.ietf-ace-oauth-params]

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-15, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-15.txt>>.

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

- [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, <<https://www.rfc-editor.org/info/rfc4279>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [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, <<https://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, <<https://www.rfc-editor.org/info/rfc7251>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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, <<https://www.rfc-editor.org/info/rfc7925>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.

- [RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <<https://www.rfc-editor.org/info/rfc8422>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

11.2. Informative References

- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <<https://www.rfc-editor.org/info/rfc6655>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

Authors' Addresses

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

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

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

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

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

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

Göran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Ludwig Seitz
Combitech
Djäknegatan 31
SE-211 35 Malmö
Sweden

Email: ludwig.seitz@combitech.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 12 May 2022

L. Seitz
Combitech
G. Selander
Ericsson
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
Arm Ltd.
8 November 2021

Authentication and Authorization for Constrained Environments (ACE)
using the OAuth 2.0 Framework (ACE-OAuth)
draft-ietf-ace-oauth-authz-46

Abstract

This specification defines a framework for authentication and authorization in Internet of Things (IoT) environments called ACE-OAuth. The framework is based on a set of building blocks including OAuth 2.0 and the Constrained Application Protocol (CoAP), thus transforming a well-known and widely used authorization solution into a form suitable for IoT devices. Existing specifications are used where possible, but extensions are added and profiles are defined to better serve the IoT use cases.

Status of This Memo

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

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

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

This Internet-Draft will expire on 12 May 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
2. Terminology	5
3. Overview	6
3.1. OAuth 2.0	7
3.2. CoAP	10
4. Protocol Interactions	11
5. Framework	14
5.1. Discovering Authorization Servers	16
5.2. Unauthorized Resource Request Message	16
5.3. AS Request Creation Hints	17
5.3.1. The Client-Nonce Parameter	19
5.4. Authorization Grants	20
5.5. Client Credentials	20
5.6. AS Authentication	21
5.7. The Authorization Endpoint	21
5.8. The Token Endpoint	21
5.8.1. Client-to-AS Request	22
5.8.2. AS-to-Client Response	25
5.8.3. Error Response	27
5.8.4. Request and Response Parameters	28
5.8.4.1. Grant Type	28
5.8.4.2. Token Type	29
5.8.4.3. Profile	29
5.8.4.4. Client-Nonce	30
5.8.5. Mapping Parameters to CBOR	30
5.9. The Introspection Endpoint	31
5.9.1. Introspection Request	32
5.9.2. Introspection Response	33
5.9.3. Error Response	34
5.9.4. Mapping Introspection Parameters to CBOR	35
5.10. The Access Token	36
5.10.1. The Authorization Information Endpoint	36

5.10.1.1.	Verifying an Access Token	38
5.10.1.2.	Protecting the Authorization Information Endpoint	39
5.10.2.	Client Requests to the RS	40
5.10.3.	Token Expiration	41
5.10.4.	Key Expiration	42
6.	Security Considerations	43
6.1.	Protecting Tokens	43
6.2.	Communication Security	44
6.3.	Long-Term Credentials	44
6.4.	Unprotected AS Request Creation Hints	45
6.5.	Minimal Security Requirements for Communication	45
6.6.	Token Freshness and Expiration	46
6.7.	Combining Profiles	47
6.8.	Unprotected Information	47
6.9.	Identifying Audiences	48
6.10.	Denial of Service Against or with Introspection	49
7.	Privacy Considerations	49
8.	IANA Considerations	50
8.1.	ACE Authorization Server Request Creation Hints	50
8.2.	CoRE Resource Type Registry	51
8.3.	OAuth Extensions Error Registration	51
8.4.	OAuth Error Code CBOR Mappings Registry	52
8.5.	OAuth Grant Type CBOR Mappings	52
8.6.	OAuth Access Token Types	53
8.7.	OAuth Access Token Type CBOR Mappings	53
8.7.1.	Initial Registry Contents	53
8.8.	ACE Profile Registry	54
8.9.	OAuth Parameter Registration	54
8.10.	OAuth Parameters CBOR Mappings Registry	54
8.11.	OAuth Introspection Response Parameter Registration	55
8.12.	OAuth Token Introspection Response CBOR Mappings Registry	56
8.13.	JSON Web Token Claims	56
8.14.	CBOR Web Token Claims	57
8.15.	Media Type Registrations	58
8.16.	CoAP Content-Format Registry	58
8.17.	Expert Review Instructions	59
9.	Acknowledgments	60
10.	References	60
10.1.	Normative References	60
10.2.	Informative References	63
Appendix A.	Design Justification	66
Appendix B.	Roles and Responsibilities	69
Appendix C.	Requirements on Profiles	71
Appendix D.	Assumptions on AS Knowledge about C and RS	72
Appendix E.	Differences to OAuth 2.0	73
Appendix F.	Deployment Examples	73

F.1. Local Token Validation	74
F.2. Introspection Aided Token Validation	78
Authors' Addresses	82

1. Introduction

Authorization is the process for granting approval to an entity to access a generic 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 can be a complex task.

While prior work on authorization solutions for the Web and for the mobile environment also applies to the Internet of Things (IoT) environment, many IoT devices are constrained, for example, in terms of processing capabilities, available memory, etc. For such devices the Constrained Application Protocol (CoAP) [RFC7252] can alleviate some resource concerns when used instead of HTTP to implement the communication flows of this specification.

Appendix A gives an overview of the constraints considered in this design, and a more detailed treatment of constraints can be found in [RFC7228]. This design aims to accommodate different IoT deployments and thus 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.

Profiles of this framework are available in separate specifications, such as [I-D.ietf-ace-dtls-authorize] or [I-D.ietf-ace-oscore-profile]. Such profiles may specify the use of the framework for a specific security protocol and the underlying transports for use in a specific deployment environment to improve interoperability. 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. More powerful devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of constrained devices. Requirements on profiles are described at contextually appropriate places throughout this specification, and also summarized in Appendix C.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

Since exchanges in this specification are described as RESTful protocol interactions, HTTP [RFC7231] offers useful terminology.

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] 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 introspection at the AS and authz-info at the RS (see Section 5.10.1 for a definition of the authz-info endpoint). The CoAP [RFC7252] definition, which is "An entity participating in the CoAP protocol" is not used in this specification.

The specifications in this document is called the "framework" or "ACE framework". When referring to "profiles of this framework" it refers to additional specifications that define the use of this specification with concrete transport and communication security protocols (e.g., CoAP over DTLS).

The term "Access Information" is used for parameters, other than the access token, provided to the client by the AS to enable it to access the RS (e.g. public key of the RS, profile supported by RS).

The term "Authorization Information" is used to denote all information, including the claims of relevant access tokens, that an RS uses to determine whether an access request should be granted.

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, other underlying protocols are not prohibited from being supported in the future, such as HTTP/2 [RFC7540], Message Queuing Telemetry Transport (MQTT) [MQTT5.0], Bluetooth Low Energy (BLE) [BLE] and QUIC [I-D.ietf-quic-transport]. Note that this document specifies protocol exchanges in terms of RESTful verbs such as GET and POST. Future profiles using protocols that do not support these verbs MUST specify how the corresponding protocol messages are transmitted instead.

A third building block is the Concise Binary Object Representation (CBOR) [RFC8949], for encodings where JSON [RFC8259] is not sufficiently compact. CBOR is a binary encoding designed for small code and message size. Self-contained tokens and protocol message payloads are encoded in CBOR when CoAP is used. When CoAP is not used, the use of CBOR remains RECOMMENDED.

A fourth building block is CBOR Object Signing and Encryption (COSE) [RFC8152], which enables object-level layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC8446]). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which are an extension to the OAuth bearer tokens. The default token format is defined in CBOR Web Token (CWT) [RFC8392]. Application-layer security for CoAP using COSE can be provided with OSCORE [RFC8613].

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 [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. It 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 the form of ACE profiles.

3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain scoped 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:

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 AS and consumed by the RS. 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.

Introspection:

Introspection is a method for a resource server or potentially a client, 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 an opaque reference rather than a self-contained token. More information about introspection in OAuth 2.0 can be found in [RFC7662].

Refresh Tokens:

Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token expires, or to obtain additional access tokens with identical or narrower scope (such access tokens may have a shorter

lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token (i.e., step (B) in Figure 1).

A refresh token in OAuth 2.0 is a string representing the authorization granted to the client by the resource owner. The string is usually opaque to the client. The token denotes an identifier used to retrieve the authorization information. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers. In this framework, refresh tokens are encoded in binary instead of strings, if used.

Proof of Possession Tokens:

A token may be bound to a cryptographic key, which is then used to bind the token to a request authorized by the token. Such tokens are called proof-of-possession tokens (or PoP tokens).

The proof-of-possession security concept used here assumes that the AS acts as a trusted third party that binds keys to tokens. In the case of 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 access token unless specifically stated otherwise.

The key bound to the token (the PoP key) may use either symmetric or asymmetric cryptography. The appropriate choice of the kind of cryptography 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 included in the token. Either the whole token or only the key MUST be encrypted in the latter case. The PoP key is also returned to client together with the token.

Asymmetric PoP key:

An asymmetric key pair is generated by 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 token and sent back to the client. The resource server consuming the token can identify the public key from the information in the token, which allows the client to use the corresponding private key for the proof of possession.

The token is either a simple reference, or a structured information object (e.g., CWT [RFC8392]) protected by a cryptographic wrapper (e.g., COSE [RFC8152]). 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 defines the use of CBOR encoding, see Section 5, for such requests and responses.

The values of the scope parameter in OAuth 2.0 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 name-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 [RFC8392] the CBOR Web Token (CWT) has been defined as an equivalent format using CBOR encoding.

The token and introspection 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).

In some deployments, a token introspection endpoint is provided by the AS, which can be used by the RS and potentially the client, if they need to request additional information regarding a received access token. The requesting entity makes a POST request to the introspection endpoint on the AS and receives information about the access token in the response. (See "Introspection" above.)

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 needs 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 into header parameters called 'options'.

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [RFC7959]. However, blockwise 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 or TLS [RFC6347][RFC8446] [I-D.ietf-tls-dtls13]. CoAP defines a number of proxy operations that require 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 at the application layer using an object-based security mechanism such as COSE [RFC8152].

One application of COSE is OSCORE [RFC8613], which provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages. In OSCORE, the CoAP messages are wrapped in COSE objects and sent using CoAP.

In this framework the use of CoAP as replacement for HTTP is RECOMMENDED for use in constrained environments. For communication security this framework does not make an explicit protocol recommendation, since the choice depends on the requirements of the

specific application. DTLS [RFC6347], [I-D.ietf-tls-dtls13] and OSCORE [RFC8613] are mentioned as examples, other protocols fulfilling the requirements from Section 6.5 are also applicable.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the token endpoint and optionally the introspection endpoint. A client obtains an access token, and optionally a refresh token, from an AS using the token endpoint and subsequently presents the access token to an RS to gain access to a protected resource. In most deployments the RS can process the access token locally, however in some cases the RS may present it to the AS via the introspection endpoint to get fresh information. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in Section 3.1.

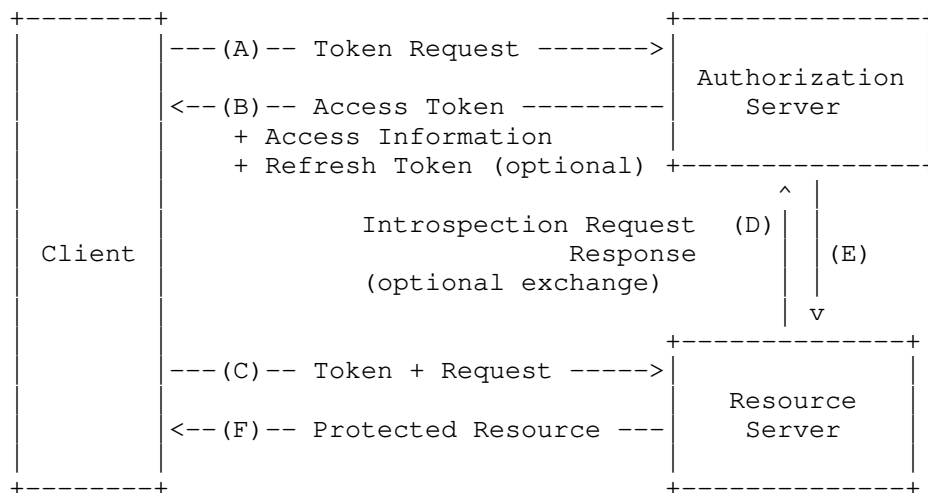


Figure 1: Basic Protocol Flow.

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 access 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 for proof-of-possession (e.g., symmetric/asymmetric cryptography or a reference to a specific key) of the access token.

Access Token Response (B):

If the request from the client has been successfully verified, authenticated, and authorized, the AS returns an access token and optionally a refresh token. Note that only certain grant types support refresh tokens. The AS can also return additional parameters, referred to as "Access Information". In addition to the response parameters defined by OAuth 2.0 and the PoP access token extension, this framework defines parameters that can be used to inform the client about capabilities of the RS, e.g. the profile the RS supports. More information about these parameters can be found in Section 5.8.4.

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 [RFC7540], QUIC [I-D.ietf-quic-transport], MQTT [MQTT5.0], Bluetooth Low Energy [BLE], 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 will 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 Access 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 Access Information. The RS verifies that the token is integrity protected and originated by the AS. It then 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.

Token Introspection Request (D):

A resource server may be configured to introspect the access token by including it in a request to the introspection endpoint at that AS. Token introspection over CoAP is defined in Section 5.9 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.

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 the communication security protocol used.

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 [RFC7521] and [RFC8628]). What grant type works best depends on the usage scenario and [RFC7744] describes many different IoT use cases but there are two grant types that cover a majority of these scenarios, namely the Authorization Code Grant (described in Section 4.1 of [RFC7521]) and the Client Credentials Grant (described in Section 4.4 of [RFC7521]). 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 [RFC8252] 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 has pre-arranged access rights for the client with the authorization server, 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. It is assumed 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 provisioning procedure implies that the client and the AS exchange 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. Confidentiality protection of the access token content would be provided on top of confidentiality protection via a communication security protocol.

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. A generic procedure is described in Section 5.1; profiles MAY define other procedures for discovery.

In Bluetooth Low Energy, for example, advertisements are broadcast 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].

5. Framework

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

Credential Provisioning

In constrained environments it cannot be assumed that the client and the RS are part of a common key infrastructure. Therefore, the AS provisions credentials and associated information to allow mutual authentication between the client and the RS. The resulting security association between the client and the RS may then also be used to bind these credentials to the access tokens the client uses.

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 [RFC8747] indicating what key is used for proof-of-possession. If a client needs to submit a new access token, e.g., to obtain additional access rights, they can request that the AS binds this token to the same key as the previous one.

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 "ace_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 if introspection is used. In constrained settings TLS is not always feasible, or desirable. Nevertheless it is REQUIRED that the communications named above are encrypted, integrity protected and protected against message replay. It is also REQUIRED that the communicating endpoints perform mutual authentication. Furthermore it MUST be assured that responses are bound to the requests in the sense that the receiver of a response can be certain that the response actually belongs to a certain request. Note that setting up such a secure communication may require some unprotected messages to be exchanged first (e.g. sending the token from the client to the RS).

Profiles MUST specify a communication security protocol between client and RS that provides the features required above. Profiles MUST specify a communication security protocol RECOMMENDED to be used between client and AS that provides the features required above. Profiles MUST specify for introspection a communication security protocol RECOMMENDED to be used between RS and AS that provides the features required above. These recommendations enable interoperability between different implementations without the need to define a new profile if the communication between C and AS, or between RS and AS, is protected with a different security protocol complying with the security requirements above.

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. When profiles of this framework use CoAP instead, it is REQUIRED to 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 CBOR encoding for a number of OAuth 2.0 parameters is specified in this document, if a profile needs to use other OAuth 2.0 parameters with CoAP it MUST specify their CBOR encoding.

Profiles that use CBOR encoding of protocol message parameters at the outermost encoding layer MUST use the content format 'application/ace+cbor'. If CoAP is used for communication, the Content-Format MUST be abbreviated with the ID: 19 (see Section 8.16).

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. If CoAP is used, it is REQUIRED to use CBOR [RFC8949] instead of JSON. Depending on the profile, the CBOR payload MAY be enclosed in a non-CBOR cryptographic wrapper.

5.1. Discovering Authorization Servers

C must discover the AS in charge of RS to determine where to request the access token. To do so, C must 1. find out the AS URI to which the token request message must be sent and 2. MUST validate that the AS with this URI is authorized to provide access tokens for this RS.

In order to determine the AS URI, 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 (see Section 5.2). How C validates the AS authorization is not in scope for this document. C may, e.g., ask its owner if this AS is authorized for this RS. C may also use a mechanism that addresses both problems at once (e.g. by querying a dedicated secure service provided by the client owner) .

5.2. Unauthorized Resource Request Message

An Unauthorized Resource Request message is a request for any resource hosted by RS for which the client does not have authorization granted. RSes MUST treat any request for a protected resource as an Unauthorized Resource Request message when any of the following hold:

- * The request has been received on an unsecured channel.
- * The RS has no valid access token for the sender of the request regarding the requested action on that resource.

- * The RS has a valid access token for the sender of the request, but that token does not authorize the requested action on the requested resource.

Note: These conditions ensure that the RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The `authz-info` endpoint, as part of the process for authorizing to protected resources, is not itself a protected resource and MUST NOT be protected as specified above (cf. Section 5.10.1).

Unauthorized Resource Request messages MUST be denied with an `"unauthorized_client"` error response. In this response, the Resource Server SHOULD provide proper `"AS Request Creation Hints"` to enable the client to request an access token from RS's AS as described in Section 5.3.

The handling of all client requests (including unauthorized ones) by the RS is described in Section 5.10.2.

5.3. AS Request Creation Hints

The `"AS Request Creation Hints"` message is sent by an RS as a response to an Unauthorized Resource Request message (see Section 5.2) to help the sender of the Unauthorized Resource Request message acquire a valid access token. The `"AS Request Creation Hints"` message is a CBOR or JSON map, with an OPTIONAL element `"AS"` specifying an absolute URI (see Section 4.3 of [RFC3986]) that identifies the appropriate AS for the RS.

The message can also contain the following OPTIONAL parameters:

- * A `"audience"` element contains an identifier the client should request at the AS, as suggested by the RS. With this parameter, when included in the access token request to the AS, the AS is able to restrict the use of access token to specific RSs. See Section 6.9 for a discussion of this parameter.
- * A `"kid"` element containing the key identifier of a key used in an existing security association between the client and the RS. The RS expects the client to request an access token bound to this key, in order to avoid having to re-establish the security association.
- * A `"cnonce"` element containing a client-nonce. See Section 5.3.1.
- * A `"scope"` element containing the suggested scope that the client should request towards the AS.

Figure 2 summarizes the parameters that may be part of the "AS Request Creation Hints".

Name	CBOR Key	Value Type
AS	1	text string
kid	2	byte string
audience	5	text string
scope	9	text or byte string
cnonce	39	byte string

Figure 2: AS Request Creation Hints

Note that the schema part of the AS parameter may need to be adapted to the security protocol that is used between the client and the AS. Thus the example AS value "coap://as.example.com/token" might need to be transformed to "coaps://as.example.com/token". It is assumed that the client can determine the correct schema part on its own depending on the way it communicates with the AS.

Figure 3 shows an example for an "AS Request Creation Hints" message payload using CBOR [RFC8949] diagnostic notation, using the parameter names instead of the CBOR keys for better human readability.

```

4.01 Unauthorized
Content-Format: application/ace+cbor
Payload :
{
  "AS" : "coaps://as.example.com/token",
  "audience" : "coaps://rs.example.com"
  "scope" : "rTempC",
  "cnonce" : h'e0a156bb3f'
}

```

Figure 3: AS Request Creation Hints payload example

In the example above, the response parameter "AS" points the receiver of this message to the URI "coaps://as.example.com/token" to request access tokens. The RS sending this response uses an internal clock that is not synchronized with the clock of the AS. Therefore, it can not reliably verify the expiration time of access tokens it receives. To ensure a certain level of access token freshness nevertheless, the RS has included a cnonce parameter (see Section 5.3.1) in the response. (The hex-sequence of the cnonce parameter is encoded in CBOR-based notation in this example.)

Figure 4 illustrates the mandatory to use binary encoding of the message payload shown in Figure 3.

```

a4                                # map(4)
 01                                # unsigned(1) (=AS)
 78 1c                            # text(28)
    636f6170733a2f2f61732e657861
    6d706c652e636f6d2f746f6b656e    # "coaps://as.example.com/token"
 05                                # unsigned(5) (=audience)
 76                                # text(22)
    636f6170733a2f2f72732e657861
    6d706c652e636f6d                # "coaps://rs.example.com"
 09                                # unsigned(9) (=scope)
 66                                # text(6)
    72546556d7043                  # "rTempC"
 18 27                            # unsigned(39) (=cnonce)
 45                                # bytes(5)
    e0a156bb3f                     #

```

Figure 4: AS Request Creation Hints example encoded in CBOR

5.3.1. The Client-Nonce Parameter

If the RS does not synchronize its clock with the AS, it could be tricked into accepting old access tokens, that are either expired or have been compromised. In order to ensure some level of token freshness in that case, the RS can use the "cnonce" (client-nonce) parameter. The processing requirements for this parameter are as follows:

- * An RS sending a "cnonce" parameter in an "AS Request Creation Hints" message MUST store information to validate that a given cnonce is fresh. How this is implemented internally is out of scope for this specification. Expiration of client-nonces should be based roughly on the time it would take a client to obtain an access token after receiving the "AS Request Creation Hints" message, with some allowance for unexpected delays.
- * A client receiving a "cnonce" parameter in an "AS Request Creation Hints" message MUST include this in the parameters when requesting an access token at the AS, using the "cnonce" parameter from Section 5.8.4.4.
- * If an AS grants an access token request containing a "cnonce" parameter, it MUST include this value in the access token, using the "cnonce" claim specified in Section 5.10.

- * An RS that is using the client-nonce mechanism and that receives an access token MUST verify that this token contains a cnonce claim, with a client-nonce value that is fresh according to the information stored at the first step above. If the cnonce claim is not present or if the cnonce claim value is not fresh, the RS MUST discard the access token. If this was an interaction with the authz-info endpoint the RS MUST also respond with an error message using a response code equivalent to the CoAP code 4.01 (Unauthorized).

5.4. Authorization Grants

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

The OAuth framework [RFC6749] 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). Further grant types have been added later, such as [RFC7521] defining an assertion-based authorization grant.

The grant type is selected depending on the use case. In cases where the client acts on behalf of the resource owner, the authorization code grant is recommended. If the client acts on behalf of the resource owner, but does not have any display or has very limited interaction possibilities, it is recommended to use the device code grant defined in [RFC8628]. In cases where the client acts autonomously the client credentials grant is recommended.

For details on the different grant types, see section 1.3 of [RFC6749]. 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.

5.5. Client Credentials

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

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

The OAuth framework defines one client credential type in section 2.3.1 of [RFC6749]: client id and client secret.

[I-D.erdman-ace-rpcc] adds raw-public-key and pre-shared-key to the client credentials types. Profiles of this framework MAY extend with an additional client credentials type using client certificates.

5.6. AS Authentication

The client credential grant 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 MUST specify how clients authenticate the AS and how communication security is implemented. By default, server side TLS certificates, as defined by OAuth 2.0, are required.

5.7. The Authorization Endpoint

The OAuth 2.0 authorization endpoint is used to interact with the resource owner and obtain an authorization grant, in certain grant flows. The primary use case for the ACE-OAuth framework is for machine-to-machine interactions that do not involve the resource owner in the authorization flow; therefore, this endpoint is out of scope here. Future profiles may define constrained adaptation mechanisms for this endpoint as well. Non-constrained clients interacting with constrained resource servers can use the specification in section 3.1 of [RFC6749] and the attack countermeasures suggested in section 4.2 of [RFC6819].

5.8. The Token Endpoint

In standard 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 the client and RS to establish shared keys or to exchange their public keys. Furthermore, this framework defines encodings using CBOR, as a substitute for JSON.

The endpoint may also be exposed over HTTPS as in classical OAuth or even other transports. A profile MUST define the details of the mapping between the fields described below, and these transports. If HTTPS is used, the semantics of Sections 4.1.3 and 4.1.4 of the OAuth 2.0 specification MUST be followed (with additions as described below). If the CoAP is some other transport with CBOR payload format is supported, the semantics described in this section MUST be followed.

For the AS to be able to issue a token, the client MUST be authenticated and present a valid grant for the scopes requested. Profiles of this framework MUST specify how the AS authenticates the client and how the communication between client and AS is protected, fulfilling the requirements specified in Section 5.

The default name of this endpoint in an url-path SHOULD be `'/token'`. However, implementations are not required to use this name and can define their own instead.

The figures of this section use CBOR diagnostic notation without the integer abbreviations for the parameters or their values for illustrative purposes. Note that implementations MUST use the integer abbreviations and the binary CBOR encoding, if the CBOR encoding is used.

5.8.1. Client-to-AS Request

The client sends a POST request to the token endpoint at the AS. The profile MUST specify how the communication is protected. The content of the request consists of the parameters specified in the relevant subsection of section 4 of the OAuth 2.0 specification [RFC6749], depending on the grant type, with the following exceptions and additions:

- * The parameter `"grant_type"` is OPTIONAL in the context of this framework (as opposed to REQUIRED in RFC6749). If that parameter is missing, the default value `"client_credentials"` is implied.
- * The `"audience"` parameter from [RFC8693] is OPTIONAL to request an access token bound to a specific audience.
- * The `"cnonce"` parameter defined in Section 5.8.4.4 is REQUIRED if the RS provided a client-nonce in the `"AS Request Creation Hints"` message Section 5.3
- * The `"scope"` parameter MAY be encoded as a byte string instead of the string encoding specified in section 3.3 of [RFC6749], in order allow compact encoding of complex scopes. The syntax of such a binary encoding is explicitly not specified here and left to profiles or applications. Note specifically that a binary encoded scope does not necessarily use the space character `'0x20'` to delimit scope-tokens.
- * The client can send an empty (null value) `"ace_profile"` parameter to indicate that it wants the AS to include the `"ace_profile"` parameter in the response. See Section 5.8.4.3.

- * A client MUST be able to use the parameters from [I-D.ietf-ace-oauth-params] in an access token request to the token endpoint and the AS MUST be able to process these additional parameters.

The default behavior, is that the AS generates a symmetric proof-of-possession key for the client. In order to use an asymmetric key pair or to re-use a key previously established with the RS, the client is supposed to use the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

If CoAP is used then these parameters MUST be provided in a CBOR map, see Figure 12.

When HTTP is used as a transport then the client makes a request to the token endpoint, the parameters MUST be encoded as defined in Appendix B of [RFC6749].

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

Figure 5 shows a request for a token with a symmetric proof-of-possession key. The content is displayed in CBOR diagnostic notation, without abbreviations for better readability.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "audience" : "tempSensor4711"
}
```

Figure 5: Example request for an access token bound to a symmetric key.

Figure 6 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example OSCORE [RFC8613] is used to provide object-security, therefore the Content-Format is "application/oscore" wrapping the "application/ace+cbor" type content. The OSCORE option has a decoded interpretation appended in parentheses for the reader's convenience. Also note that in this example the audience is implicitly known by both client and AS. Furthermore note that this example uses the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
OSCORE: 0x09, 0x05, 0x44, 0x6C
      (h=0, k=1, n=001, partialIV= 0x05, kid=[0x44, 0x6C])
Content-Format: "application/oscore"
Payload:
  0x44025d1 ... (full payload omitted for brevity) ... 68b3825e
```

Decrypted payload:

```
{
  "client_id" : "myclient",
  "req_cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h'11',
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+lrb7btKLv8EX4'
    }
  }
}
```

Figure 6: Example token request bound to an asymmetric key.

Figure 7 shows a request for a token where a previously communicated proof-of-possession key is only referenced using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "audience" : "valve424",
  "scope" : "read",
  "req_cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

Figure 7: Example request for an access token bound to a key reference.

Refresh tokens are typically not stored as securely as proof-of-possession keys in requesting clients. Proof-of-possession based refresh token requests **MUST NOT** request different proof-of-possession keys or different audiences in token requests. Refresh token requests can only use to request access tokens bound to the same proof-of-possession key and the same audience as access tokens issued in the initial token request.

5.8.2. 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 response code equivalent to 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 5.8.3.

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]. If the client has requested a specific proof-of-possession key using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params], this may also influence which profile the AS selects, as it needs to support the use of the key type requested the client.

The content of the successful reply is the Access Information. When using CoAP, the payload **MUST** be encoded as a CBOR map, when using HTTP the encoding is a JSON map as specified in section 5.1 of [RFC6749]. In both cases the parameters specified in Section 5.1 of [RFC6749] are used, with the following additions and changes:

ace_profile:

OPTIONAL unless the request included an empty ace_profile parameter in which case it is MANDATORY. This indicates the profile that the client **MUST** use towards the RS. See Section 5.8.4.3 for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

token_type:

This parameter is OPTIONAL, as opposed to 'required' in [RFC6749]. 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.

Furthermore [I-D.ietf-ace-oauth-params] defines additional parameters that the AS MUST be able to use when responding to a request to the token endpoint.

Figure 8 summarizes the parameters that can currently be part of the Access Information. Future extensions may define additional parameters.

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
error	RFC 6749
error_description	RFC 6749
error_uri	RFC 6749
ace_profile	[this document]
cnf	[I-D.ietf-ace-oauth-params]
rs_cnf	[I-D.ietf-ace-oauth-params]

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key, which is defined in [I-D.ietf-ace-oauth-params]. Note that the key identifier 'kid' is only used to simplify indexing and retrieving the key, and no assumptions should be made that it is unique in the domains of either the client or the RS.


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

Figure 9: Example AS response with an access token bound to a symmetric key.

5.8.3. Error Response

The error responses for interactions with the AS are generally equivalent to the ones defined in Section 5.2 of [RFC6749], with the following exceptions:

- * When using CoAP the payload MUST be encoded as a CBOR map, with the Content-Format "application/ace+cbor". When using HTTP the payload is encoded in JSON as specified in section 5.2 of [RFC6749].
- * A response code equivalent to the CoAP code 4.00 (Bad Request) MUST be used for all error responses, except for `invalid_client` where a response code equivalent to the CoAP 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" MUST be abbreviated using the codes specified in Figure 12, when a CBOR encoding is used.
- * The error code (i.e., value of the "error" parameter) MUST be abbreviated as specified in Figure 10, when a CBOR encoding is used.

Name	CBOR Values	Original Specification
invalid_request	1	section 5.2 of [RFC6749]
invalid_client	2	section 5.2 of [RFC6749]
invalid_grant	3	section 5.2 of [RFC6749]
unauthorized_client	4	section 5.2 of [RFC6749]
unsupported_grant_type	5	section 5.2 of [RFC6749]
invalid_scope	6	section 5.2 of [RFC6749]
unsupported_pop_key	7	[this document]
incompatible_ace_profiles	8	[this document]

Figure 10: CBOR abbreviations for common error codes

In addition to the error responses defined in OAuth 2.0, the following behavior MUST be implemented by the AS:

- * If the client submits an asymmetric key in the token request that the RS cannot process, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "unsupported_pop_key" specified in Figure 10.
- * If the client and the RS it has requested an access token for do not share a common profile, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "incompatible_ace_profiles" specified in Figure 10.

5.8.4. 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.

5.8.4.1. Grant Type

The abbreviations specified in the registry defined in Section 8.5 MUST be used in CBOR encodings instead of the string values defined in [RFC6749], if CBOR payloads are used.

Name	CBOR Value	Original Specification
password	0	s. 4.3.2 of [RFC6749]
authorization_code	1	s. 4.1.3 of [RFC6749]
client_credentials	2	s. 4.4.2 of [RFC6749]
refresh_token	3	s. 6 of [RFC6749]

Figure 11: CBOR abbreviations for common grant types

5.8.4.2. Token Type

The "token_type" parameter, defined in section 5.1 of [RFC6749], allows 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 by the client to the RS is performed MUST be specified by the profiles.

The values in the "token_type" parameter MUST use the CBOR abbreviations defined in the registry specified by Section 8.7, if a CBOR encoding is used.

In this framework the "pop" value for the "token_type" parameter is the default. The AS may, however, provide a different value from those registered in [IANA.OAuthAccessTokenTypes].

5.8.4.3. Profile

Profiles of this framework MUST define the communication protocol and the communication security protocol between the client and the RS. The security protocol MUST provide encryption, integrity and replay protection. It MUST also provide a binding between requests and responses. Furthermore profiles MUST define a list of allowed proof-of-possession methods, if they support proof-of-possession tokens.

A profile MUST specify an identifier that MUST be used to uniquely identify itself in the "ace_profile" parameter. The textual representation of the profile identifier is intended for human readability and for JSON-based interactions, it MUST NOT be used for CBOR-based interactions. Profiles MUST register their identifier in the registry defined in Section 8.8.

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

Clients that want the AS to provide them with the "ace_profile" parameter in the access token response can indicate that by sending a ace_profile parameter with a null value for CBOR-based interactions, or an empty string if CBOR is not used, in the access token request.

5.8.4.4. Client-Nonce

This parameter MUST be sent from the client to the AS, if it previously received a "cnonce" parameter in the "AS Request Creation Hints" Section 5.3. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (base64url without padding encoded binary [RFC4648]) if CBOR is not used. It MUST copy the value from the cnonce parameter in the "AS Request Creation Hints".

5.8.5. Mapping Parameters to CBOR

If CBOR encoding is used, all OAuth parameters in access token requests and responses MUST be mapped to CBOR types as specified in the registry defined by Section 8.10, using the given integer abbreviation for the map keys.

Note that we have aligned the abbreviations corresponding to claims with the abbreviations defined in [RFC8392].

Note also that abbreviations from -24 to 23 have a 1 byte encoding size in CBOR. We have thus chosen to assign abbreviations in that range to parameters we expect to be used most frequently in constrained scenarios.

Name	CBOR Key	Value Type	Original Specification
access_token	1	byte string	[RFC6749]
expires_in	2	unsigned integer	[RFC6749]
audience	5	text string	[RFC8693]
scope	9	text or byte string	[RFC6749]
client_id	24	text string	[RFC6749]
client_secret	25	byte string	[RFC6749]
response_type	26	text string	[RFC6749]
redirect_uri	27	text string	[RFC6749]
state	28	text string	[RFC6749]
code	29	byte string	[RFC6749]
error	30	integer	[RFC6749]
error_description	31	text string	[RFC6749]
error_uri	32	text string	[RFC6749]
grant_type	33	unsigned integer	[RFC6749]
token_type	34	integer	[RFC6749]
username	35	text string	[RFC6749]
password	36	text string	[RFC6749]
refresh_token	37	byte string	[RFC6749]
ace_profile	38	integer	[this document]
cnonce	39	byte string	[this document]

Figure 12: CBOR mappings used in token requests and responses

5.9. The Introspection Endpoint

Token introspection [RFC7662] MAY be implemented by the AS, and the RS. When implemented, it MAY be used by the RS and to query the AS for metadata about a given token, e.g., validity or scope. Analogous to the protocol defined in [RFC7662] for HTTP and JSON, this section defines adaptations to more constrained environments using CBOR and leaving the choice of the application protocol to the profile.

Communication between the requesting entity and the introspection endpoint at the AS MUST be integrity protected and encrypted. The communication security protocol MUST also provide a binding between requests and responses. Furthermore, the two interacting parties MUST perform mutual authentication. Finally, the AS SHOULD verify that the requesting entity 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 the requesting entity and the AS is implemented.

The default name of this endpoint in an url-path SHOULD be `"/introspect"`. However, implementations are not required to use this name and can define their own instead.

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

5.9.1. Introspection Request

The requesting entity sends a POST request to the introspection endpoint at the AS. The profile MUST specify how the communication is protected. If CoAP is used, the payload MUST be encoded as a CBOR map with a "token" entry containing the access token. Further optional parameters representing additional context that is known by the requesting entity to aid the AS in its response MAY be included.

For CoAP-based interaction, all messages MUST use the content type `"application/ace+cbor"`. For HTTP the encoding defined in section 2.1 of [RFC7662] is used.

The same parameters are required and optional as in Section 2.1 of [RFC7662].

For example, Figure 13 shows an RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that object security based on OSCORE [RFC8613] is assumed in this example, therefore the Content-Format is `"application/oscore"`. Figure 14 shows the decoded payload.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "introspect"
OSCORE: 0x09, 0x05, 0x25
Content-Format: "application/oscore"
Payload:
... COSE content ...
```

Figure 13: Example introspection request.

```
{
  "token" : b64'7gj0dXJQ43U',
  "token_type_hint" : "PoP"
}
```

Figure 14: Decoded payload.

5.9.2. Introspection Response

If the introspection request is authorized and successfully processed, the AS sends a response with the response code equivalent to the CoAP 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 5.9.3.

In a successful response, the AS encodes the response parameters in a map. If CoAP is used, this MUST be encoded as a CBOR map, if HTTP is used the JSON encoding specified in section 2.2 of [RFC7662] is used. The map containing the response payload includes the same required and optional parameters as in Section 2.2 of [RFC7662] with the following additions:

`ace_profile` OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.8.4.3 for more details on the formatting of this parameter. If this parameter is absent, the AS assumes that the RS implicitly knows which profile to use towards the client.

`cnonce` OPTIONAL. A client-nonce provided to the AS by the client. The RS MUST verify that this corresponds to the client-nonce previously provided to the client in the "AS Request Creation Hints". See Section 5.3 and Section 5.8.4.4. Its value is a byte string when encoded in CBOR and the base64url encoding of this byte string without padding when encoded in JSON [RFC4648].

`cti` OPTIONAL. The "cti" claim associated to this access token. This parameter has the same meaning and processing rules as the "jti" parameter defined in section 3.1.2 of [RFC7662] except that its value is a byte string when encoded in CBOR and the base64url encoding of this byte string without padding when encoded in JSON [RFC4648].

`exp` OPTIONAL. The "expires-in" claim associated to this access token. See Section 5.10.3.

Furthermore [I-D.ietf-ace-oauth-params] defines more parameters that the AS MUST be able to use when responding to a request to the introspection endpoint.

For example, Figure 15 shows an AS response to the introspection request in Figure 13. Note that this example contains the "cnf" parameter defined in [I-D.ietf-ace-oauth-params].

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "active" : true,
  "scope" : "read",
  "ace_profile" : "coap_dtls",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'39Gqlw',
      "k" : b64'hJtXhkV8FJG+Onbc6mxCcQh'
    }
  }
}
```

Figure 15: Example introspection response.

5.9.3. 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 and CoAP is used the payload MUST be encoded as a CBOR map and the Content-Format "application/ace+cbor" MUST be used. For HTTP the encoding defined in section 2.3 of [RFC6749] is used.
- * If the credentials used by the requesting entity (usually the RS) are invalid the AS MUST respond with the response code equivalent to the CoAP code 4.01 (Unauthorized) and use the required and optional parameters from Section 2.3 in [RFC7662].
- * If the requesting entity does not have the right to perform this introspection request, the AS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden). In this case no payload is returned.
- * The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12.
- * The error codes MUST be abbreviated using the codes specified in the registry defined by Section 8.4.

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".

5.9.4. Mapping Introspection Parameters to CBOR

If CBOR is used, the introspection request and response parameters **MUST** be mapped to CBOR types as specified in the registry defined by Section 8.12, using the given integer abbreviation for the map key.

Note that we have aligned abbreviations that correspond to a claim with the abbreviations defined in [RFC8392] and the abbreviations of parameters with the same name from Section 5.8.5.

Parameter name	CBOR Key	Value Type	Original Specification
iss	1	text string	[RFC7662]
sub	2	text string	[RFC7662]
aud	3	text string	[RFC7662]
exp	4	integer or floating-point number	[RFC7662]
nbf	5	integer or floating-point number	[RFC7662]
iat	6	integer or floating-point number	[RFC7662]
cti	7	byte string	[this document]
scope	9	text or byte string	[RFC7662]
active	10	True or False	[RFC7662]
token	11	byte string	[RFC7662]
client_id	24	text string	[RFC7662]
error	30	integer	[RFC7662]
error_description	31	text string	[RFC7662]
error_uri	32	text string	[RFC7662]
token_type_hint	33	text string	[RFC7662]
token_type	34	integer	[RFC7662]
username	35	text string	[RFC7662]
ace_profile	38	integer	[this document]
cnonce	39	byte string	[this document]
exi	40	unsigned integer	[this document]

Figure 16: CBOR mappings for Token Introspection Parameters.

5.10. The Access Token

In this framework the use of CBOR Web Token (CWT) as specified in [RFC8392] is RECOMMENDED.

In order to facilitate offline processing of access tokens, this document uses the "cnf" claim from [RFC8747] and the "scope" claim from [RFC8693] for JWT- and CWT-encoded tokens. In addition to string encoding specified for the "scope" claim, a binary encoding MAY be used. The syntax of such an encoding is explicitly not specified here and left to profiles or applications, specifically note that a binary encoded scope does not necessarily use the space character '0x20' to delimit scope-tokens.

If the AS needs to convey a hint to the RS about which profile it should use to communicate with the client, the AS MAY include an "ace_profile" claim in the access token, with the same syntax and semantics as defined in Section 5.8.4.3.

If the client submitted a client-nonce parameter in the access token request Section 5.8.4.4, the AS MUST include the value of this parameter in the "cnonce" claim specified here. The "cnonce" claim uses binary encoding.

5.10.1. The Authorization Information Endpoint

The access token, containing authorization information and information about the proof-of-possession method 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 a RESTful protocol such as 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 the authz-info endpoint at the RS with the access token in the payload. The CoAP Content-Format or HTTP Media Type MUST reflect the format of the token, e.g. application/cwt for CBOR Web Tokens, if no Content-Format or Media Type is defined for the token format, application/octet-stream MUST be used.

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 a response code equivalent to CoAP's 2.01 (Created). Section 5.10.1.1 outlines how an RS MUST proceed to verify the validity of an access token.

The RS MUST be prepared to store at least one access token for future use. This is a difference to how access tokens are handled in OAuth 2.0, where the access token is typically sent along with each request, and therefore not stored at the RS.

When using this framework it is RECOMMENDED that an RS stores only one token per proof-of-possession key. This means that an additional token linked to the same key will supersede any existing token at the RS, by replacing the corresponding authorization information. The reason is that this greatly simplifies (constrained) implementations, with respect to required storage and resolving a request to the applicable token. The use of multiple access tokens for a single client increases the strain on the resource server as it must consider every access token and calculate the actual permissions of the client. Also, tokens may contradict each other which may lead the server to enforce wrong permissions. If one of the access tokens expires earlier than others, the resulting permissions may offer insufficient protection.

If the payload sent to the authz-info endpoint does not parse to a token, the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request).

The RS MAY make an introspection request to validate the token before responding to the POST request to the authz-info endpoint, e.g. if the token is an opaque reference. Some transport protocols may provide a way to indicate that the RS is busy and the client should retry after an interval; this type of status update would be appropriate while the RS is waiting for an introspection response.

Profiles MUST specify whether the authz-info endpoint is protected, including whether error responses from this endpoint are 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 default name of this endpoint in an url-path is '/authz-info', however implementations are not required to use this name and can define their own instead.

5.10.1.1. Verifying an Access Token

When an RS receives an access token, it MUST verify it before storing it. The details of token verification depends on various aspects, including the token encoding, the type of token, the security protection applied to the token, and the claims. The token encoding matters since the security protection differs between the token encodings. For example, a CWT token uses COSE while a JWT token uses JOSE. The type of token also has an influence on the verification procedure since tokens may be self-contained whereby token verification may happen locally at the RS while a token-by-reference requires further interaction with the authorization server, for example using token introspection, to obtain the claims associated with the token reference. Self-contained tokens MUST, at least be integrity protected but they MAY also be encrypted.

For self-contained tokens the RS MUST process the security protection of the token first, as specified by the respective token format. For CWT the description can be found in [RFC8392] and for JWT the relevant specification is [RFC7519]. This MUST include a verification that security protection (and thus the token) was generated by an AS that has the right to issue access tokens for this RS.

In case the token is communicated by reference the RS needs to obtain the claims first. When the RS uses token introspection the relevant specification is [RFC7662] with CoAP transport specified in Section 5.9.

Errors may happen during this initial processing stage:

- * If the verification of the security wrapper fails, or the token was issued by an AS that does not have the right to issue tokens for the receiving RS, the RS MUST discard the token and, if this was an interaction with authz-info, return an error message with a response code equivalent to the CoAP code 4.01 (Unauthorized).
- * If the claims cannot be obtained the RS MUST discard the token and, in case of an interaction via the authz-info endpoint, return an error message with a response code equivalent to the CoAP code 4.00 (Bad Request).

Next, the RS MUST verify claims, if present, contained in the access token. Errors are returned when claim checks fail, in the order of priority of this list:

iss The issuer claim (if present) must identify the AS that has

produced the security protection for the access token. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.01 (Unauthorized).

exp The expiration date must be in the future. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info the RS MUST also respond with a response code equivalent to the CoAP code 4.01 (Unauthorized). Note that the RS has to terminate access rights to the protected resources at the time when the tokens expire.

aud The audience claim must refer to an audience that the RS identifies with. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.03 (Forbidden).

scope The RS must recognize value of the scope claim. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.00 (Bad Request). The RS MAY provide additional information in the error response, to clarify what went wrong.

Additional processing may be needed for other claims in a way specific to a profile or the underlying application.

Note that the Subject (sub) claim cannot always be verified when the token is submitted to the RS since the client may not have authenticated yet. Also note that a counter for the expires_in (exp) claim MUST be initialized when the RS first verifies this token.

Also note that profiles of this framework may define access token transport mechanisms that do not allow for error responses. Therefore the error messages specified here only apply if the token was sent to the authz-info endpoint.

When sending error responses, the RS MAY use the error codes from Section 3.1 of [RFC6750], to provide additional details to the client.

5.10.1.2. Protecting the Authorization Information Endpoint

As this framework can be used in RESTful environments, it is important to make sure that attackers cannot perform unauthorized requests on the authz-info endpoints, other than submitting access tokens.

Specifically it SHOULD NOT be possible to perform GET, DELETE or PUT on the authz-info endpoint.

The RS SHOULD implement rate limiting measures to mitigate attacks aiming to overload the processing capacity of the RS by repeatedly submitting tokens. For CoAP-based communication the RS could use the mechanisms from [RFC8516] to indicate that it is overloaded.

5.10.2. Client Requests to the RS

Before sending a request to an RS, the client MUST verify that the keys used to protect this communication are still valid. See Section 5.10.4 for details on how the client determines the validity of the keys used.

If an RS receives a request from a client, and the target resource requires authorization, the RS MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession binding that token to the request.

The response code MUST be 4.01 (Unauthorized) in case the client has not performed the proof-of-possession, or if RS has no valid access token for the client. If RS has an access token for the client but the token does not authorize access for the resource that was requested, RS MUST reject the request with a 4.03 (Forbidden). If RS has an access token for the client but it does not cover the action that was 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.

Note: The RS MAY use introspection for timely validation of an access token, at the time when a request is presented.

Note: Matching the claims of the access token (e.g., scope) to a specific request is application specific.

If the request matches a valid token and the client has performed the proof-of-possession for that token, the RS continues to process the request as specified by the underlying application.

5.10.3. Token Expiration

Depending on the capabilities of the RS, there are various ways in which it can verify the expiration of a received access token. Here follows a list of the possibilities including what functionality they require of the RS.

- * The token is a CWT and includes an "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 specification.
- * The RS verifies the validity of the token by performing an introspection request as specified in Section 5.9. 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 RS to AS).
- * In order to support token expiration for devices that have no reliable way of synchronizing their internal clocks, this specification defines the following approach: The claim "exp" ("expires in") can be used, to provide the RS with the lifetime of the token in seconds from the time the RS first receives the token. This mechanism only works for self-contained tokens, i.e. CWTs and JWTs. For CWTs this parameter is encoded as unsigned integer, while JWTs encode this as JSON number.
- * Processing this claim requires that the RS does the following:
 - For each token the RS receives, that contains an "exp" claim: Keep track of the time it received that token and revisit that list regularly to expunge expired tokens.
 - Keep track of the identifiers of tokens containing the "exp" claim that have expired (in order to avoid accepting them again). In order to avoid an unbounded memory usage growth, this MUST be implemented in the following way when the "exp" claim is used:
 - o When creating the token, the AS MUST add a 'cti' claim (or 'jti' for JWTs) to the access token. The value of this claim MUST be created as the binary representation of the concatenation of the identifier of the RS with a sequence number counting the tokens containing an 'exp' claim, issued by this AS for the RS.

- o The RS MUST store the highest sequence number of an expired token containing the "exp" claim that it has seen, and treat tokens with lower sequence numbers as expired. Note that this could lead to discarding valid tokens with lower sequence numbers, if the AS were to issue tokens of different validity time for the same RS. The assumption is that typically tokens in such a scenario would all have the same validity time.

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

5.10.4. Key Expiration

The AS provides the client with key material that the RS uses. This can either be a common symmetric PoP-key, or an asymmetric key used by the RS to authenticate towards the client. Since there is currently no expiration metadata associated to those keys, the client has no way of knowing if these keys are still valid. This may lead to situations where the client sends requests containing sensitive information to the RS using a key that is expired and possibly in the hands of an attacker, or accepts responses from the RS that are not properly protected and could possibly have been forged by an attacker.

In order to prevent this, the client must assume that those keys are only valid as long as the related access token is. Since the access token is opaque to the client, one of the following methods MUST be used to inform the client about the validity of an access token:

- * The client knows a default validity time for all tokens it is using (i.e. how long a token is valid after being issued). This information could be provisioned to the client when it is registered at the AS, or published by the AS in a way that the client can query.
- * The AS informs the client about the token validity using the "expires_in" parameter in the Access Information.

A client that is not able to obtain information about the expiration of a token MUST NOT use this token.

6. Security Considerations

Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work. Furthermore [RFC6819] provides additional security considerations for OAuth which apply to IoT deployments as well. If the introspection endpoint is used, the security considerations from [RFC7662] also apply.

The following subsections address issues specific to this document and it's use in constrained environments.

6.1. Protecting Tokens

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 (MAC) or an Authenticated Encryption with Associated Data (AEAD) algorithm. 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 used for proof-of-possession. If the access token contains the symmetric key, this symmetric key **MUST** be encrypted by the authorization server so that only the resource server can decrypt it. Note that using an AEAD algorithm is preferable over using a MAC unless the token needs to be publicly readable.

If the token is intended for multiple recipients (i.e. an audience that is a group), integrity protection of the token with a symmetric key, shared between the AS and the recipients, is not sufficient, since any of the recipients could modify the token undetected by the other recipients. Therefore a token with a multi-recipient audience **MUST** be protected with an asymmetric signature.

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. The same shared secret **MUST NOT** be used as proof-of-possession key with multiple resource servers since the benefit from using the proof-of-possession concept is then significantly reduced.

If clients are capable of doing so, they should frequently request fresh access tokens, as this allows the AS to keep the lifetime of the tokens short. This allows the AS to use shorter proof-of-possession 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 permission.

In certain situations it may be necessary to revoke an access token that is still valid. Client-initiated revocation is specified in [RFC7009] for OAuth 2.0. Other revocation mechanisms are currently not specified, as the underlying assumption in OAuth is that access tokens are issued with a relatively short lifetime. This may not hold true for disconnected constrained devices, needing access tokens with relatively long lifetimes, and would therefore necessitate further standardization work that is out of scope for this document.

6.2. Communication Security

Communication with the authorization server MUST use confidentiality protection. This step is extremely important since the client or the RS may obtain the proof-of-possession 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. The requirements for communication security of profiles are specified in Section 5.

Additional protection for the access token can be applied by encrypting it, for example encryption of CWTs is specified in Section 5.1 of [RFC8392]. Such additional protection can be necessary if the token is later transferred over an insecure connection (e.g. when it is sent to the authz-info endpoint).

Care must be taken by developers to prevent leakage of the PoP credentials (i.e., the private key or the symmetric key). An adversary in possession of the PoP 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 may get physical access to the devices and can therefore use physical extraction techniques to gain access to memory contents. This risk can be mitigated to some extent by making sure that keys are refreshed frequently, by using software isolation techniques and by using hardware security.

6.3. Long-Term Credentials

Both clients and RSs have long-term credentials that are used to secure communications, and authenticate to the AS. These credentials need to be protected against unauthorized access. In constrained devices, deployed in publicly accessible places, such protection can be difficult to achieve without specialized hardware (e.g. secure key storage memory).

If credentials are lost or compromised, the operator of the affected devices needs to have procedures to invalidate any access these credentials give and to revoke tokens linked to such credentials. The loss of a credential linked to a specific device MUST NOT lead to a compromise of other credentials not linked to that device, therefore secret keys used for authentication MUST NOT be shared between more than two parties.

Operators of clients or RS SHOULD have procedures in place to replace credentials that are suspected to have been compromised or that have been lost.

Operators also SHOULD have procedures for decommissioning devices, that include securely erasing credentials and other security critical material in the devices being decommissioned.

6.4. Unprotected AS Request Creation Hints

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the "AS Request Creation Hints" contained in an unprotected response from RS to an unauthorized request (see Section 5.3) are authentic. C therefore MUST determine if an AS is authorized to provide access tokens for a certain RS. How this determination is implemented is out of scope for this document and left to the applications.

6.5. Minimal Security Requirements for Communication

This section summarizes the minimal requirements for the communication security of the different protocol interactions.

C-AS All communication between the client and the Authorization Server MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the client MUST be bound to the client's request to avoid attacks where the attacker swaps the intended response for an older one valid for a previous request. This requires that the client and the Authorization Server have previously exchanged either a shared secret or their public keys in order to negotiate a secure communication. Furthermore the client MUST be able to determine whether an AS has the authority to issue access tokens for a certain RS. This can for example be done through pre-configured lists, or through an online lookup mechanism that in turn also must be secured.

RS-AS The communication between the Resource Server and the Authorization Server via the introspection endpoint MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the RS MUST be bound to the RS's request. This

requires that the RS and the Authorization Server have previously exchanged either a shared secret, or their public keys in order to negotiate a secure communication. Furthermore the RS MUST be able to determine whether an AS has the authority to issue access tokens itself. This is usually configured out of band, but could also be performed through an online lookup mechanism provided that it is also secured in the same way.

C-RS The initial communication between the client and the Resource Server can not be secured in general, since the RS is not in possession of an access token for that client, which would carry the necessary parameters. If both parties support DTLS without client authentication it is RECOMMEND to use this mechanism for protecting the initial communication. After the client has successfully transmitted the access token to the RS, a secure communication protocol MUST be established between client and RS for the actual resource request. This protocol MUST provide confidentiality, integrity and replay protection as well as a binding between requests and responses. This requires that the client learned either the RS's public key or received a symmetric proof-of-possession key bound to the access token from the AS. The RS must have learned either the client's public key or a shared symmetric key from the claims in the token or an introspection request. Since ACE does not provide profile negotiation between C and RS, the client MUST have learned what profile the RS supports (e.g. from the AS or pre-configured) and initiate the communication accordingly.

6.6. Token Freshness and Expiration

An RS that is offline faces the problem of clock drift. Since it cannot synchronize its clock with the AS, it may be tricked into accepting old access tokens that are no longer valid or have been compromised. In order to prevent this, an RS may use the nonce-based mechanism (cnonce) defined in Section 5.3 to ensure freshness of an Access Token subsequently presented to this RS.

Another problem with clock drift is that evaluating the standard token expiration claim "exp" can give unpredictable results.

Acceptable ranges of clock drift are highly dependent on the concrete application. Important factors are how long access tokens are valid, and how critical timely expiration of access token is.

The expiration mechanism implemented by the "exp" claim, based on the first time the RS sees the token was defined to provide a more predictable alternative. The "exp" approach has some drawbacks that need to be considered:

A malicious client may hold back tokens with the "exi" claim in order to prolong their lifespan.

If an RS loses state (e.g. due to an unscheduled reboot), it may lose the current values of counters tracking the "exi" claims of tokens it is storing.

The first drawback is inherent to the deployment scenario and the "exi" solution. It can therefore not be mitigated without requiring the RS be online at times. The second drawback can be mitigated by regularly storing the value of "exi" counters to persistent memory.

6.7. Combining Profiles

There may be use cases where different transport and security protocols are allowed for the different interactions, and, if that is not explicitly covered by an existing profile, it corresponds to combining profiles into a new one. For example, a new profile could specify that a previously-defined MQTT-TLS profile is used between the client and the RS in combination with a previously-defined CoAP-DTLS profile for interactions between the client and the AS. The new profile that combines existing profiles MUST specify how the existing profiles' security properties are achieved. Any profile therefore MUST clearly specify its security requirements and MUST document if its security depends on the combination of various protocol interactions.

6.8. Unprotected Information

Communication with the authz-info endpoint, as well as the various error responses defined in this framework, all potentially include sending information over an unprotected channel. These messages may leak information to an adversary, or may be manipulated by active attackers to induce incorrect behavior. For example error responses for requests to the Authorization Information endpoint can reveal information about an otherwise opaque access token to an adversary who has intercepted this token.

As far as error messages are concerned, this framework is written under the assumption that, in general, the benefits of detailed error messages outweigh the risk due to information leakage. For particular use cases, where this assessment does not apply, detailed error messages can be replaced by more generic ones.

In some scenarios it may be possible to protect the communication with the authz-info endpoint (e.g. through DTLS with only server-side authentication). In cases where this is not possible, it is RECOMMENDED to use encrypted CWTs or tokens that are opaque references and need to be subjected to introspection by the RS.

If the initial unauthorized resource request message (see Section 5.2) is used, the client MUST make sure that it is not sending sensitive content in this request. While GET and DELETE requests only reveal the target URI of the resource, POST and PUT requests would reveal the whole payload of the intended operation.

Since the client is not authenticated at the point when it is submitting an access token to the authz-info endpoint, attackers may be pretending to be a client and trying to trick an RS to use an obsolete profile that in turn specifies a vulnerable security mechanism via the authz-info endpoint. Such an attack would require a valid access token containing an "ace_profile" claim requesting the use of said obsolete profile. Resource Owners should update the configuration of their RS's to prevent them from using such obsolete profiles.

6.9. Identifying Audiences

The audience claim as defined in [RFC7519] and the equivalent "audience" parameter from [RFC8693] are intentionally vague on how to match the audience value to a specific RS. This is intended to allow application specific semantics to be used. This section attempts to give some general guidance for the use of audiences in constrained environments.

URLs are not a good way of identifying mobile devices that can switch networks and thus be associated with new URLs. If the audience represents a single RS, and asymmetric keys are used, the RS can be uniquely identified by a hash of its public key. If this approach is used it is RECOMMENDED to apply the procedure from section 3 of [RFC6920].

If the audience addresses a group of resource servers, the mapping of group identifier to individual RS has to be provisioned to each RS before the group-audience is usable. Managing dynamic groups could be an issue, if any RS is not always reachable when the groups' memberships change. Furthermore, issuing access tokens bound to symmetric proof-of-possession keys that apply to a group-audience is problematic, as an RS that is in possession of the access token can impersonate the client towards the other RSs that are part of the group. It is therefore NOT RECOMMENDED to issue access tokens bound to a group audience and symmetric proof-of possession keys.

Even the client must be able to determine the correct values to put into the "audience" parameter, in order to obtain a token for the intended RS. Errors in this process can lead to the client inadvertently obtaining a token for the wrong RS. The correct values for "audience" can either be provisioned to the client as part of its configuration, or dynamically looked up by the client in some directory. In the latter case the integrity and correctness of the directory data must be assured. Note that the "audience" hint provided by the RS as part of the "AS Request Creation Hints" Section 5.3 is not typically source authenticated and integrity protected, and should therefore not be treated a trusted value.

6.10. Denial of Service Against or with Introspection

The optional introspection mechanism provided by OAuth and supported in the ACE framework allows for two types of attacks that need to be considered by implementers.

First, an attacker could perform a denial of service attack against the introspection endpoint at the AS in order to prevent validation of access tokens. To maintain the security of the system, an RS that is configured to use introspection **MUST NOT** allow access based on a token for which it couldn't reach the introspection endpoint.

Second, an attacker could use the fact that an RS performs introspection to perform a denial of service attack against that RS by repeatedly sending tokens to its authz-info endpoint that require an introspection call. RS can mitigate such attacks by implementing rate limits on how many introspection requests they perform in a given time interval for a certain client IP address submitting tokens to /authz-info. When that limit has been reached, incoming requests from that address are rejected for a certain amount of time. A general rate limit on the introspection requests should also be considered, to mitigate distributed attacks.

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 and 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 this can be prevented by the Resource Owner. To do so, the resource owner needs to bind the grants it issues to anonymous, ephemeral credentials that do not allow the AS to link different grants and thus different access token requests by the same client.

The claims contained in a token can reveal privacy sensitive information about the client and the RS to any party having access to them (whether by processing the content of a self-contained token or by introspection). The AS SHOULD be configured to minimize the information about clients and RSs disclosed in the tokens it issues.

If 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 the token may, e.g., reveal the audience, the scope and the confirmation method used by the client. The latter may reveal the identity of the device or application running the client. This may be linkable to the identity of the person using the client (if there is a person and not a machine-to-machine interaction).

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 RSs. A set of colluding RSs or an attacker able to obtain the access tokens will be able to link the requests, or even to determine the client's identity.

An unprotected response to an unauthorized request (see Section 5.3) 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. Even the absolute URI of the AS may reveal sensitive information about the service that RS provides. Developers must ensure that the RS does not disclose information that has an impact on the privacy of the stakeholders in the "AS Request Creation Hints". They may choose to use a different mechanism for the discovery of the AS if necessary. If means of encrypting communication between C and RS already exist, more detailed information may be included with an error response to provide C with sufficient information to react on that particular error.

8. IANA Considerations

This document creates several registries with a registration policy of "Expert Review"; guidelines to the experts are given in Section 8.17.

8.1. ACE Authorization Server Request Creation Hints

This specification establishes the IANA "ACE Authorization Server Request Creation Hints" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of the registry are:

Name The name of the parameter

CBOR Key CBOR map key for the parameter. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Value Type The CBOR data types allowable for the values of this parameter.

Reference This contains a pointer to the public specification of the request creation hint abbreviation, if one exists.

This registry will be initially populated by the values in Figure 2. The Reference column for all of these entries will be this document.

8.2. CoRE Resource Type Registry

IANA is requested to register a new Resource Type (rt=) Link Target Attribute in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" [IANA.CoreParameters] registry:

- * Value: ace.ai
- * Description: ACE-OAuth authz-info endpoint resource.
- * Reference: [this document]

Specific ACE-OAuth profiles can use this common resource type for defining their profile-specific discovery processes.

8.3. OAuth Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry [IANA.OAuthExtensionsErrorRegistry].

- * Error name: unsupported_pop_key
- * Error usage location: token error response
- * Related protocol extension: [this document]
- * Change Controller: IETF
- * Specification document(s): Section 5.8.3 of [this document]

- * Error name: incompatible_ace_profiles
- * Error usage location: token error response

- * Related protocol extension: [this document]
- * Change Controller: IETF
- * Specification document(s): Section 5.8.3 of [this document]

8.4. OAuth Error Code CBOR Mappings Registry

This specification establishes the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of the registry are:

Name	The OAuth Error Code name, refers to the name in Section 5.2. of [RFC6749], e.g., "invalid_request".
CBOR Value	CBOR abbreviation for this error code. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the error code abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the error code, if one exists.

This registry will be initially populated by the values in Figure 10. The Reference column for all of these entries will be this document.

8.5. OAuth Grant Type CBOR Mappings

This specification establishes the IANA "OAuth Grant Type CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The name of the grant type as specified in Section 1.3 of [RFC6749].
CBOR Value	CBOR abbreviation for this grant type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the grant type abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the grant type, if one exists.

This registry will be initially populated by the values in Figure 11. The Reference column for all of these entries will be this document.

8.6. OAuth Access Token Types

This section registers the following new token type in the "OAuth Access Token Types" registry [IANA.OAuthAccessTokenTypes].

- * Type name: PoP
- * Additional Token Endpoint Response Parameters: "cnf", "rs_cnf" see section 3.1 of [RFC8747] and section 3.1 of [I-D.ietf-ace-oauth-params].
- * HTTP Authentication Scheme(s): N/A
- * Change Controller: IETF
- * Specification document(s): [this document]

8.7. OAuth Access Token Type CBOR Mappings

This specification established the IANA "OAuth Access Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The name of token type as registered in the OAuth Access Token Types registry, e.g., "Bearer".
CBOR Value	CBOR abbreviation for this token type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference	This contains a pointer to the public specification of the OAuth token type abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of the OAuth token type, if one exists.

8.7.1. Initial Registry Contents

- * Name: Bearer
- * Value: 1
- * Reference: [this document]
- * Original Specification: [RFC6749]

- * Name: PoP
- * Value: 2
- * Reference: [this document]
- * Original Specification: [this document]

8.8. ACE Profile Registry

This specification establishes the IANA "ACE Profile" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this registry are:

Name The name of the profile, to be used as value of the profile attribute.

Description Text giving an overview of the profile and the context it is developed for.

CBOR Value CBOR abbreviation for this profile name. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as "Expert Review". Integer values less than -65536 are marked as Private Use.

Reference This contains a pointer to the public specification of the profile abbreviation, if one exists.

This registry will be initially empty and will be populated by the registrations from the ACE framework profiles.

8.9. OAuth Parameter Registration

This specification registers the following parameter in the "OAuth Parameters" registry [IANA.OAuthParameters]:

- * Name: ace_profile
- * Parameter Usage Location: token response
- * Change Controller: IETF
- * Reference: Section 5.8.2 and Section 5.8.4.3 of [this document]

8.10. OAuth Parameters CBOR Mappings Registry

This specification establishes the IANA "OAuth Parameters CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

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

CBOR Key CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].

Value Type The allowable CBOR data types for values of this parameter.

Reference This contains a pointer to the public specification of the OAuth parameter abbreviation, if one exists.

Original Specification This contains a pointer to the public specification of the OAuth parameter, if one exists.

This registry will be initially populated by the values in Figure 12. The Reference column for all of these entries will be this document.

8.11. OAuth Introspection Response Parameter Registration

This specification registers the following parameters in the OAuth Token Introspection Response registry [IANA.TokenIntrospectionResponse].

- * **Name:** ace_profile
- * **Description:** The ACE profile used between client and RS.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** cnonce
- * **Description:** "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** cti
- * **Description:** "CWT ID". The identifier of a CWT as defined in [RFC8392].
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

- * **Name:** exp
- * **Description:** "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker form of token expiration for devices that cannot synchronize their internal clocks.
- * **Change Controller:** IETF
- * **Reference:** Section 5.9.2 of [this document]

8.12. OAuth Token Introspection Response CBOR Mappings Registry

This specification establishes the IANA "OAuth Token Introspection Response CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name	The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".
CBOR Key	CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Value Type	The allowable CBOR data types for values of this parameter.
Reference	This contains a pointer to the public specification of the introspection response parameter abbreviation, if one exists.
Original Specification	This contains a pointer to the public specification of OAuth Token Introspection parameter, if one exists.

This registry will be initially populated by the values in Figure 16. The Reference column for all of these entries will be this document.

Note that the mappings of parameters corresponding to claim names intentionally coincide with the CWT claim name mappings from [RFC8392].

8.13. JSON Web Token Claims

This specification registers the following new claims in the JSON Web Token (JWT) registry of JSON Web Token Claims [IANA.JsonWebTokenClaims]:

- * Claim Name: ace_profile
- * Claim Description: The ACE profile a token is supposed to be used with.
- * Change Controller: IETF
- * Reference: Section 5.10 of [this document]

- * Claim Name: cnonce
- * Claim Description: "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- * Change Controller: IETF
- * Reference: Section 5.10 of [this document]

- * Claim Name: `exp`
- * Claim Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker form of token expiration for devices that cannot synchronize their internal clocks.
- * Change Controller: IETF
- * Reference: Section 5.10.3 of [this document]

8.14. CBOR Web Token Claims

This specification registers the following new claims in the "CBOR Web Token (CWT) Claims" registry [IANA.CborWebTokenClaims].

- * Claim Name: `ace_profile`
- * Claim Description: The ACE profile a token is supposed to be used with.
- * JWT Claim Name: `ace_profile`
- * Claim Key: TBD (suggested: 38)
- * Claim Value Type(s): integer
- * Change Controller: IETF
- * Specification Document(s): Section 5.10 of [this document]

- * Claim Name: `cnonce`
- * Claim Description: The client-nonce sent to the AS by the RS via the client.
- * JWT Claim Name: `cnonce`
- * Claim Key: TBD (suggested: 39)
- * Claim Value Type(s): byte string
- * Change Controller: IETF
- * Specification Document(s): Section 5.10 of [this document]

- * Claim Name: `exp`
- * Claim Description: The expiration time of a token measured from when it was received at the RS in seconds.
- * JWT Claim Name: `exp`
- * Claim Key: TBD (suggested: 40)
- * Claim Value Type(s): integer
- * Change Controller: IETF
- * Specification Document(s): Section 5.10.3 of [this document]

- * Claim Name: `scope`
- * Claim Description: The scope of an access token as defined in [RFC6749].
- * JWT Claim Name: `scope`
- * Claim Key: TBD (suggested: 9)
- * Claim Value Type(s): byte string or text string
- * Change Controller: IETF
- * Specification Document(s): Section 4.2 of [RFC8693]

8.15. Media Type Registrations

This specification registers the 'application/ace+cbor' media type for messages of the protocols defined in this document carrying parameters encoded in CBOR. This registration follows the procedures specified in [RFC6838].

Type name: application

Subtype name: ace+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: Must be encoded as CBOR map containing the protocol parameters defined in [this document].

Security considerations: See Section 6 of [this document]

Interoperability considerations: N/A

Published specification: [this document]

Applications that use this media type: The type is used by authorization servers, clients and resource servers that support the ACE framework with CBOR encoding as specified in [this document].

Fragment identifier considerations: N/A

Additional information: N/A

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

Intended usage: COMMON

Restrictions on usage: none

Author: Ludwig Seitz <ludwig.seitz@combitech.se>

Change controller: IETF

8.16. CoAP Content-Format Registry

This specification registers the following entry to the "CoAP Content-Formats" registry:

Media Type: application/ace+cbor

Encoding: -

ID: TBD (suggested: 19)

Reference: [this document]

8.17. Expert Review Instructions

All of the IANA registries established in this document are defined to use a registration policy of Expert Review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- * Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered, and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.
- * Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- * Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on.
- * Since a high degree of overlap is expected between these registries and the contents of the OAuth parameters [IANA.OAuthParameters] registries, experts should require new registrations to maintain alignment with parameters from OAuth that have comparable functionality. Deviation from this alignment should only be allowed if there are functional differences, that are motivated by the use case and that cannot be easily or efficiently addressed by comparable OAuth parameters.

9. Acknowledgments

This document is a product of the ACE working group of the IETF.

Thanks to 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.

Thanks to the authors of draft-ietf-oauth-pop-key-distribution, from where parts of the security considerations were copied.

Thanks to Stefanie Gerdes, Olaf Bergmann, and Carsten Bormann for contributing their work on AS discovery from draft-gerdes-ace-dcaf-authorize (see Section 5.1) and the considerations on multiple access tokens.

Thanks to Jim Schaad and Mike Jones for their comprehensive reviews.

Thanks to Benjamin Kaduk for his input on various questions related to this work.

Thanks to Cigdem Sengul for some very useful review comments.

Thanks to Carsten Bormann for contributing the text for the CoRE Resource Type registry.

Thanks to Roman Danyliw for suggesting the Appendix E (including its contents).

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova. Ludwig Seitz was also received further funding for this work by Vinnova in the context of the CelticNext project Critisec.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-params]

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-params-16, 7 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-params-16.txt>>.

- [IANA.CborWebTokenClaims]
IANA, "CBOR Web Token (CWT) Claims",
<<https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>>.
- [IANA.CoreParameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>>.
- [IANA.JsonWebTokenClaims]
IANA, "JSON Web Token Claims",
<<https://www.iana.org/assignments/jwt/jwt.xhtml#claims>>.
- [IANA.OAuthAccessTokenTypes]
IANA, "OAuth Access Token Types",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-types>>.
- [IANA.OAuthExtensionsErrorRegistry]
IANA, "OAuth Extensions Error Registry",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#extensions-error>>.
- [IANA.OAuthParameters]
IANA, "OAuth Parameters",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#parameters>>.
- [IANA.TokenIntrospectionResponse]
IANA, "OAuth Token Introspection Response",
<<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-introspection-response>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <<https://www.rfc-editor.org/info/rfc6750>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <<https://www.rfc-editor.org/info/rfc7662>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <<https://www.rfc-editor.org/info/rfc8693>>.
- [RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <<https://www.rfc-editor.org/info/rfc8747>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

10.2. Informative References

- [BLE] Bluetooth SIG, "Bluetooth Core Specification v5.1", Section 4.4, January 2019, <<https://www.bluetooth.com/specifications/bluetooth-core-specification/>>.
- [I-D.erdtdman-ace-rpcc]
Seitz, L. and S. Erdtman, "Raw-Public-Key and Pre-Shared-Key as OAuth client credentials", Work in Progress, Internet-Draft, draft-erdtdman-ace-rpcc-02, 30 October 2017, <<https://www.ietf.org/archive/id/draft-erdtdman-ace-rpcc-02.txt>>.
- [I-D.ietf-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)", Work in Progress, Internet-Draft, draft-ietf-ace-dtls-authorize-18, 4 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-dtls-authorize-18.txt>>.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>>.

[I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-quic-transport-34.txt>>.

[I-D.ietf-tls-dtls13]

Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.

[Margil0impact]

Margi, C. B., de Oliveira, B.T., de Sousa, G.T., Simplicio Jr, M.A., Barreto, P.S.L.M., Carvalho, T.C.M.B., Naeslund, M., and R. Gold, "Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds", Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), August 2010.

[MQTT5.0] Banks, A., Briggs, E., Borgendale, K., and R. Gupta, "MQTT Version 5.0", OASIS Standard, March 2019, <<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>>.

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

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

[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, <<https://www.rfc-editor.org/info/rfc6819>>.

- [RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <<https://www.rfc-editor.org/info/rfc7009>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [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, <<https://www.rfc-editor.org/info/rfc7521>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [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, <<https://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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <<https://www.rfc-editor.org/info/rfc8252>>.

- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <<https://www.rfc-editor.org/info/rfc8414>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8516] Keranen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.
- [RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <<https://www.rfc-editor.org/info/rfc8628>>.

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 (roughly by a factor of 10 compared to AES) [Margil10impact]. 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 allow, 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 of performing, 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 allow, 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 only a small amount of RAM and flash memory, which places limitations on 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. An authentication and key establishment protocol, e.g., the DTLS handshake, in comparison with assisted key establishment, also has an impact on memory and code footprints.

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers may 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. Deployments making use of CoAP are expected, but this framework is not limited to them. Other protocols such as HTTP, or even protocols such as Bluetooth Smart communication that do not necessarily use IP, could also be used. The latter raises the need for application-layer security over the various interfaces.

In the light of these constraints we have made the following design decisions:

CBOR, COSE, CWT:

When using this framework, it is RECOMMENDED to use CBOR [RFC8949] as data format. Where CBOR data needs to be protected, the use of COSE [RFC8152] is RECOMMENDED. Furthermore, where self-contained tokens are needed, it is RECOMMENDED to use of CWT [RFC8392]. These measures aim at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

CoAP:

When using this framework, it is RECOMMENDED to use of CoAP [RFC7252] instead of HTTP. This does not preclude the use of other protocols specifically aimed at constrained devices, like, e.g., Bluetooth Low Energy (see Section 3.2). This aims again at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

Access Information:

This framework defines the name "Access Information" for data concerning the RS that the AS returns to the client in an access token response (see Section 5.8.2). This aims at enabling scenarios where a powerful client, supporting multiple profiles, needs to interact with an RS for which it does not know the supported profiles and the raw public key.

Proof-of-Possession:

This framework makes use of proof-of-possession tokens, using the "cnf" claim [RFC8747]. A request parameter "cnf" and a Response parameter "cnf", both having a value space semantically and syntactically identical to the "cnf" claim, are defined for the token endpoint, to allow requesting and stating confirmation keys. This aims at making token theft harder. Token theft is specifically relevant in constrained use cases, as communication often passes through middle-boxes, which could be able to steal bearer tokens and use them to gain unauthorized access.

Authz-Info endpoint:

This framework introduces a new way of providing access tokens to an RS by exposing a authz-info endpoint, to which access tokens can be POSTed. This aims at reducing the size of the request message and the code complexity at the RS. The size of the request message is problematic, since many constrained protocols have severe message size limitations at the physical layer (e.g., in the order of 100 bytes). This means that larger packets get fragmented, which in turn combines badly with the high rate of packet loss, and the need to retransmit the whole message if one packet gets lost. Thus separating sending of the request and sending of the access tokens helps to reduce fragmentation.

Client Credentials Grant:

In this framework the use of the client credentials grant is RECOMMENDED for machine-to-machine communication use cases, where manual intervention of the resource owner to produce a grant token is not feasible. The intention is that the resource owner would instead pre-arrange authorization with the AS, based on the client's own credentials. The client can then (without manual intervention) obtain access tokens from the AS.

Introspection:

In this framework the use of access token introspection is RECOMMENDED in cases where the client is constrained in a way that it can not easily obtain new access tokens (i.e. it has connectivity issues that prevent it from communicating with the AS). In that case it is RECOMMENDED to use a long-term token, that could be a simple reference. The RS is assumed to be able to communicate with the AS, and can therefore perform introspection, in order to learn the claims associated with the token reference. The advantage of such an approach is that the resource owner can change the claims associated to the token reference without having to be in contact with the client, thus granting or revoking access rights.

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_type, 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 that 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 the RS and manage corresponding security contexts.
- * Register clients and authentication credentials.
- * Allow Resource Owners to configure and update access control policies related to their registered RSs.
- * Expose the token endpoint to allow clients to request tokens.
- * Authenticate clients that wish to request a token.
- * Process a token request using 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 RSs that wish to get an introspection response.
- * If providing an introspection endpoint: Process token introspection requests.
- * Optionally: Handle token revocation.
- * Optionally: Provide discovery metadata. See [RFC8414]
- * Optionally: Handle refresh tokens.

Client

- * Discover the AS in charge of the RS that is to be targeted with a request.
- * Submit the token request (see step (A) of Figure 1).
 - Authenticate to 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 keys (rpk) or certificates are used, make sure the AS has the right rpk or certificate for this client.
- * Process the access token and Access Information (see step (B) of Figure 1).
 - Check that the Access Information provides the necessary security parameters (e.g., PoP key, information on communication security protocols supported by the RS).
 - Safely store the proof-of-possession key.
 - If provided by the AS: Safely store the refresh token.
- * Send the token and request to the RS (see step (C) of Figure 1).
 - 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 (see step (F) of Figure 1) of the RS.
- 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 a recognized AS.
 - Check the token's integrity.
 - Verify that the token applies to this RS.
 - Check that the token has not expired (if the token provides expiration information).
 - Store the token so that it can be retrieved in the context of a matching request.
- Note: The order proposed here is not normative, any process that arrives at an equivalent result can be used. A noteworthy consideration is whether one can use cheap operations early on to quickly discard non-applicable or invalid tokens, before performing expensive cryptographic operations (e.g. doing an expiration check before verifying a signature).
- * 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 mechanism(s).
 - * Safely store credentials such as raw public keys for authentication or proof-of-possession keys linked to access tokens.

Appendix C. Requirements on Profiles

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

- * Optionally define new methods for the client to discover the necessary permissions and AS for accessing a resource, different from the one proposed in Section 5.1. Section 4
- * Optionally specify new grant types. Section 5.4

- * Optionally define the use of client certificates as client credential type. Section 5.5
- * Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.8.4.3
- * Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS). This must provide encryption, integrity and replay protection. Section 5.8.4.3
- * Specify how the client and the RS mutually authenticate. 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 5.8.4.2
- * Specify a unique `ace_profile` identifier. Section 5.8.4.3
- * If introspection is supported: Specify the communication and security protocol for introspection. Section 5.9
- * Specify the communication and security protocol for interactions between client and AS. This must provide encryption, integrity protection, replay protection and a binding between requests and responses. Section 5 and Section 5.8
- * Specify how/if the `authz-info` endpoint is protected, including how error responses are protected. Section 5.10.1
- * Optionally define other methods of token transport than the `authz-info` endpoint. Section 5.10.1

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 an RS in order to be able to respond to requests to the token and introspection 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) that the RS supports.
- * 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 and AS (if any).
- * The symmetric key shared between RS and AS (if any).
- * The raw public key of the client or RS (if any).

- * Whether the RS has synchronized time (and thus is able to use the 'exp' claim) or not.

Appendix E. Differences to OAuth 2.0

This document adapts OAuth 2.0 to be suitable for constrained environments. This section lists the main differences from the normative requirements of OAuth 2.0.

- * Use of TLS -- 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 if introspection is used. This framework requires similar security properties, but does not require that they be realized with TLS. See Section 5.
- * Cardinality of "grant_type" parameter -- In client-to-AS requests using OAuth 2.0, the "grant_type" parameter is required (per [RFC6749]). In this framework, this parameter is optional. See Section 5.8.1.
- * Encoding of "scope" parameter -- In client-to-AS requests using OAuth 2.0, the "scope" parameter is string encoded (per [RFC6749]). In this framework, this parameter may also be encoded as a byte string. See Section 5.8.1.
- * Cardinality of "token_type" parameter -- in AS-to-client responses using OAuth 2.0, the token_type parameter is required (per [RFC6749]). In this framework, this parameter is optional. See Section 5.8.2.
- * Access token retention -- in OAuth 2.0, the access token may be sent with every request to the RS. The exact use of access tokens depends on the semantics of the application and the session management concept it uses. In this framework, the RS must be able to store these tokens for later use. See Section 5.10.1.

Appendix F. 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 an RS is performed. This requires the security of the requests and responses between the clients and the RS to be considered.

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

F.1. Local Token Validation

In this scenario, the case where the resource server is offline is considered, 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 17.

A: The client first generates a public-private key pair used for communication security with the RS. The client sends a CoAP POST request to the token endpoint at the AS. The security of this request can be transport or application layer. It is up to the communication security profile to define. In the example it is assumed that both client and AS have performed mutual authentication e.g. via DTLS. 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 2.05 Content response containing the Access Information, including the access token. The PoP access token contains the public key of the client, and the Access 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", in order to mitigate attacks using stolen client credentials. 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 it is assumed 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.

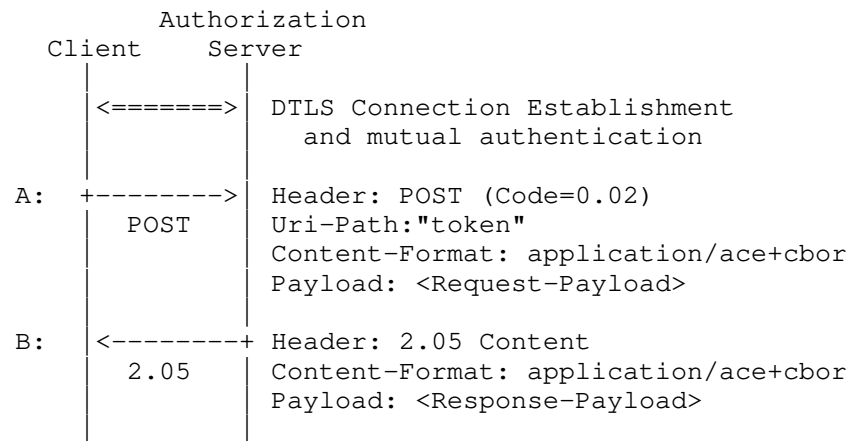


Figure 17: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 18 Note that the parameter "rs_cnf" from [I-D.ietf-ace-oauth-params] is used to inform the client about the resource server's public key.

```

Request-Payload :
{
  "audience" : "tempSensorInLivingRoom",
  "client_id" : "myclient",
  "req_cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

Response-Payload :
{
  "access_token" : b64'0INDoQEkoQVnKkXfb7xaWqMTf6 ...',
  "rs_cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'MKBCTNIcKUSDii1lySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y" : b64'4Et16SRW2YiLUrN5vfvVHuhp7x8Px1tmWWlbbM4IFyM'
    }
  }
}

```

Figure 18: Request and Response Payload Details.

The content of the access token is shown in Figure 19.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1563451500",
  "exp" : "1563453000",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f830J3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

```

Figure 19: Access Token including Public Key of the client.

Messages C and F are shown in Figure 20 - Figure 21.

C: The client then sends the PoP access token to the authz-info endpoint at the RS. This is a plain CoAP POST request, i.e., no transport or application-layer security is used between client and RS since the token is integrity protected between the AS and RS. The RS verifies that the PoP access token was created by a known and trusted AS, that it applies to this RS, and that it is valid. The RS caches the security context together with authorization information about this client contained in the PoP access token.

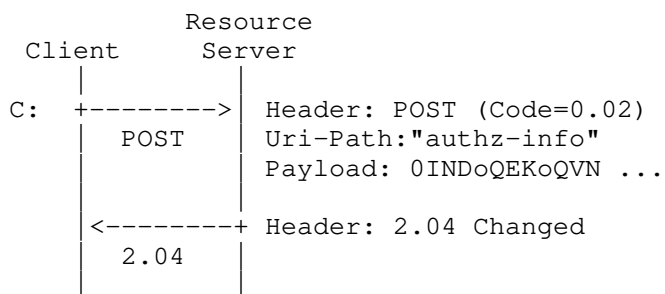


Figure 20: Access Token provisioning to RS

The client and the RS runs the DTLS handshake using the raw public keys established in step B and C.

The client sends a CoAP GET request to /temperature on RS over DTLS. The RS verifies that the request is authorized, based on previously established security context.

F: The RS responds over the same DTLS channel with a CoAP 2.05 Content response, containing a resource representation as payload.

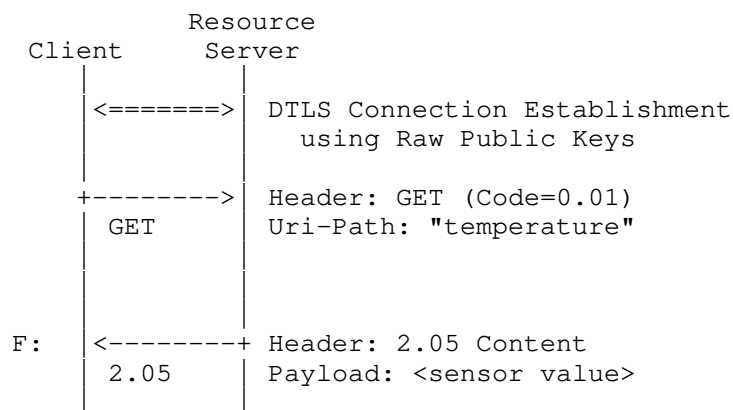


Figure 21: Resource Request and Response protected by DTLS.

F.2. Introspection Aided Token Validation

In this deployment scenario it is assumed that a client is not able to access the AS at the time of the access request, whereas the RS is assumed to be connected to the back-end infrastructure. Thus the RS 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. Since the client is constrained, the token will not be self contained (i.e. not a CWT) but instead just a reference. The resource server uses its connectivity to learn about the claims associated to the access token by using introspection, which is shown in the example below.

In the example interactions between an offline client (key fob), an RS (online lock), and an AS is shown. It is assumed 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 22.

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 be used to authorize the key for his car at the car manufacturer's 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 sent at the time of access. So the scope and audience parameters are set quite wide to start with, while tailored values narrowing down the claims to the specific RS being accessed can be provided to that RS during an introspection step.

A: The client sends a CoAP POST request to the token endpoint at AS. The request contains the Audience parameter set to "PACS1337" (PACS, Physical Access System), a value that identifies the physical access control system to which the individual doors are connected. The AS generates an access token as an opaque string, which it can match to the specific client and the targeted audience. It furthermore generates a symmetric proof-of-

possession key. The communication security and authentication between client and AS is assumed to have been provided at transport layer (e.g. via DTLS) using a pre-shared security context (psk, rpk or certificate).

B: The AS responds with a CoAP 2.05 Content response, containing as payload the Access Information, including the access token and the symmetric proof-of-possession key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key is used as the PreSharedKey.

Note: In this example we are using a symmetric key for a multi-RS audience, which is not recommended normally (see Section 6.9). However in this case the risk is deemed to be acceptable, since all the doors are part of the same physical access control system, and therefore the risk of a malicious RS impersonating the client towards another RS is low.

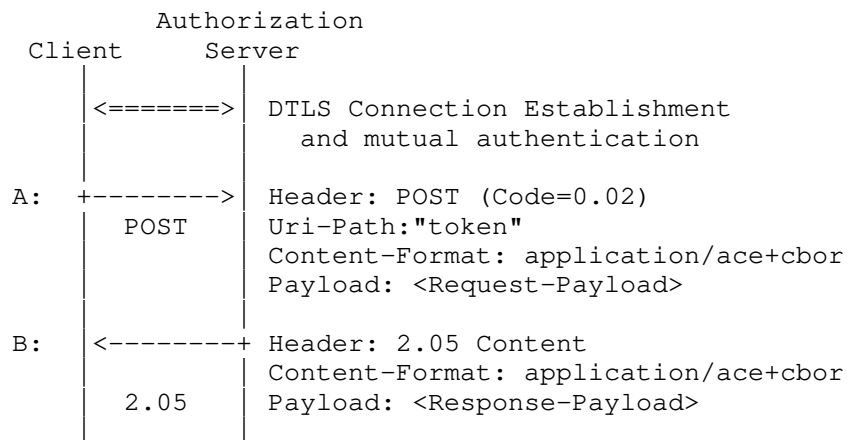


Figure 22: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 23.

```

Request-Payload:
{
  "client_id" : "keyfob",
  "audience" : "PACS1337"
}

Response-Payload:
{
  "access_token" : b64'VGZzdCB0b2t1bG==',
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "oct",
      "alg" : "HS256",
      "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
    }
  }
}

```

Figure 23: Request and Response Payload for C offline

The access token in this case is just an opaque byte 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 sends the token to the introspection endpoint on the AS using a CoAP POST request. In this example RS and AS are assumed to have performed mutual authentication using a pre shared security context (psk, rpki or certificate) with the RS acting as DTLS client.

E: The AS provides the introspection response (2.05 Content) 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. Note that our example in Figure 25 assumes a pre-established key (e.g. one used by the client and the RS for a previous token) that is now only referenced by its key-identifier 'kid'.

After receiving message E, the RS responds to the client's POST in step C with the CoAP response code 2.01 (Created).

```

          Resource
      Client  Server
      |      |
C: +----->| Header: POST (T=CON, Code=0.02)

```

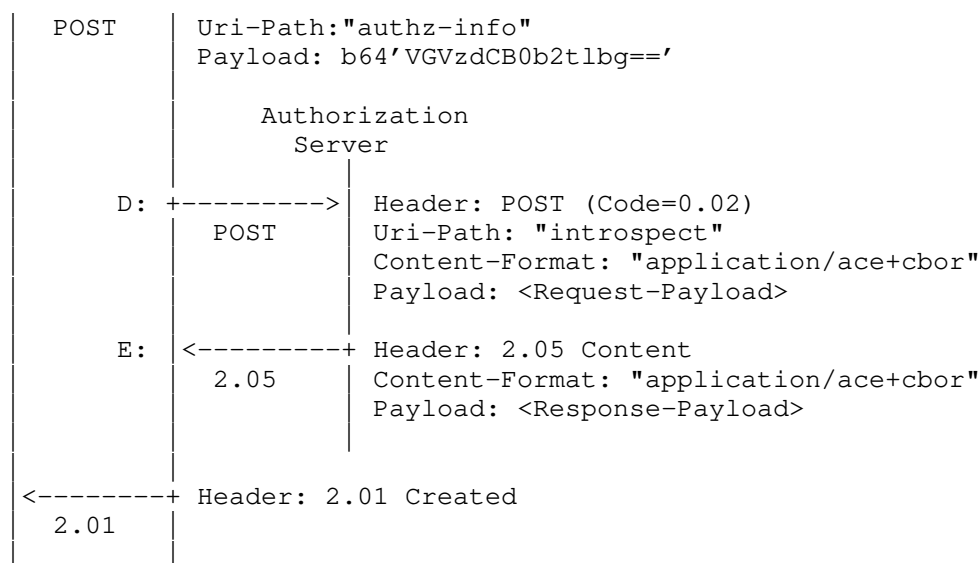


Figure 24: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 25.

Request-Payload:

```

{
  "token" : b64'VGZzdCB0b2t1bg==' ,
  "client_id" : "FrontDoor",
}

```

Response-Payload:

```

{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1563454000,
  "cnf" : {
    "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk'
  }
}

```

Figure 25: 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 the RS, changing the state of the door to locked.

F: The RS responds with a appropriate over the secure DTLS channel.

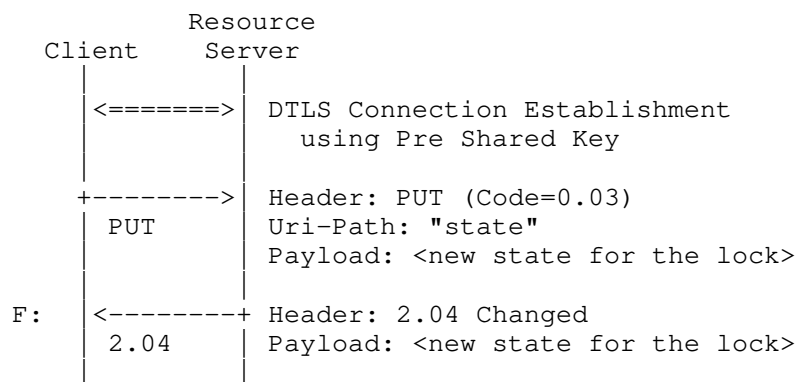


Figure 26: Resource request and response protected by OSCORE

Authors' Addresses

Ludwig Seitz
 Combitech
 Djäknegatan 31
 SE-211 35 Malmö
 Sweden

Email: ludwig.seitz@combitech.com

Goeran Selander
 Ericsson
 Faroegatan 6
 SE-164 80 Kista
 Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
 Sweden

Email: erik@wahlstromstekniska.se

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

Email: erdtman@spotify.com

Hannes Tschofenig
Arm Ltd.
6067 Absam
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2018

L. Seitz
RISE SICS AB
F. Palombini
Ericsson AB
M. Gunnarsson
RISE SICS AB
October 25, 2017

OSCORE profile of the Authentication and Authorization for Constrained
Environments Framework
draft-seitz-ace-oscoap-profile-06

Abstract

This memo specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework. It utilizes Object Security for Constrained RESTful Environments (OSCORE) to provide communication security, server authentication, and proof-of-possession for a key owned by the client and bound to an OAuth 2.0 access token.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 28, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Client to Resource Server	3
2.1. Signaling the use of OSCORE	3
2.2. Key establishment for OSCORE	4
3. Client to Authorization Server	6
4. Resource Server to Authorization Server	7
5. Security Considerations	7
6. Privacy Considerations	7
7. IANA Considerations	7
8. References	7
8.1. Normative References	7
8.2. Informative References	8
Appendix A. Profile Requirements	9
Appendix B. Using the pop-key with EDHOC (EDHOC+OSCORE)	10
B.1. Using Asymmetric Keys	10
B.2. Using Symmetric Keys	12
B.3. Processing	13
Acknowledgments	15
Authors' Addresses	16

1. Introduction

This memo specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use CoAP [RFC7252] 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. In order to provide communication security, proof of possession, and server authentication they use Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security]. Optionally the client and the resource server may also use CoAP and OSCORE to communicate with the authorization server.

OSCORE specifies how to use CBOR Object Signing and Encryption (COSE) [RFC8152] to secure CoAP messages. In order to provide replay and reordering protection OSCORE also introduces sequence numbers that are used together with COSE.

Note that OSCORE can be used to secure CoAP messages, as well as HTTP and combinations of HTTP and CoAP; a profile of ACE similar to the one described in this document, with the difference of using HTTP instead of CoAP as communication protocol, could be specified analogously to this one.

1.1. Terminology

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

Certain security-related terms 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). It is assumed in this document that a given resource on a specific RS is associated to a unique AS.

2. Client to Resource Server

The use of OSCORE for arbitrary CoAP messages is specified in [I-D.ietf-core-object-security]. This section defines the specific uses and their purpose for securing the communication between a client and a resource server, and the parameters needed to negotiate the use of this profile with the token resource at the authorization server as specified in section 5.5 of the ACE framework [I-D.ietf-ace-oauth-authz].

2.1. Signaling the use of OSCORE

A client requests a token at an AS via the /token resource. This follows the message formats specified in section 5.5.1 of the ACE framework [I-D.ietf-ace-oauth-authz].

The AS responding to a successful access token request as defined in section 5.5.2 of the ACE framework can signal that the use of OSCORE is REQUIRED for a specific access token by including the "profile" parameter with the value "coap_oscore" in the access token response. This means that the client MUST use OSCORE towards all resource

servers for which this access token is valid, and follow Section 2.2 to derive the security context to run OSCORE.

The error response procedures defined in section 5.5.3 of the ACE framework are unchanged by this profile.

Note the the client and the authorization server MAY OPTIONALLY use OSCORE to protect the interaction via the /token resource. See Section 3 for details.

2.2. Key establishment for OSCORE

Section 3.2 of OSCORE [I-D.ietf-core-object-security] defines how to derive a security context based on a shared master secret and a set of other parameters, established between client and server. The proof-of-possession key (pop-key) provisioned from the AS MAY, in case of pre-shared keys, be used directly as master secret in OSCORE.

If OSCORE is used directly with the symmetric pop-key as master secret, then the AS MUST provision the following data, in response to the access token request:

- o a master secret
- o the sender identifier
- o the recipient identifier

Additionally, the AS MAY provision the following data, in the same response. In case these parameters are omitted, the default values are used as described in section 3.2 of [I-D.ietf-core-object-security].

- o an AEAD algorithm
- o a KDF algorithm
- o a salt
- o a replay window type and size

The master secret MUST be communicated as COSE_Key in the 'cnf' parameter of the access token response as defined in section 5.5.4.5 of [I-D.ietf-ace-oauth-authz]. The AEAD algorithm MAY be included as the 'alg' parameter in the COSE_Key; the KDF algorithm MAY be included as the 'kdf' parameter of the COSE_Key and the salt MAY be included as the 'slt' parameter of the COSE_Key as defined in table 1. The same parameters MUST be included as metadata of the access

token; if the token is a CWT [I-D.ietf-ace-cbor-web-token], the same COSE_Key structure MUST be placed in the 'cnf' claim of this token. The AS MUST also assign identifiers to both client and RS, which are then used as Sender ID and Recipient ID in the OSCORE context as described in section 3.1 of [I-D.ietf-core-object-security]. These identifiers MUST be unique in the set of all clients and RS identifiers for a certain AS. Moreover, these MUST be included in the COSE_Key as header parameters, as defined in table 1.

We assume in this document that a resource is associated to one single AS, which makes it possible to assume unique identifiers for each client requesting a particular resource to a RS. If this is not the case, collisions of identifiers may appear in the RS, in which case the RS needs to have a mechanism in place to disambiguate identifiers or mitigate their effect.

Note that C should set the Sender ID of its security context to the clientId value received and the Recipient ID to the serverId value, and RS should do the opposite.

name	label	CBOR type	registry	description
clientId	TBD	bstr		Identifies the client in an OSCORE context using this key
serverId	TBD	bstr		Identifies the server in an OSCORE context using this key
kdf	TBD	bstr		Identifies the KDF algorithm in an OSCORE context using this key
slt	TBD	bstr		Identifies the master salt in an OSCORE context using this key

Table 1: Additional common header parameters for COSE_Key

Figure 1 shows an example of such an AS response, in CBOR diagnostic notation without the tag and value abbreviations.

```

Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
    (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "alg" : "AES-CCM-16-64-128",
      "clientId" : b64'qA',
      "serverId" : b64'Qg',
      "k" : b64'+aDg2jjU+eIiOFca9lObw'
    }
  }
}

```

Figure 1: Example AS response with OSCORE parameters.

Figure 2 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```

{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "alg" : "AES-CCM-16-64-128",
      "clientId" : b64'Qg',
      "serverId" : b64'qA',
      "k" : b64'+aDg2jjU+eIiOFca9lObw'
    }
  }
}

```

Figure 2: Example CWT with OSCORE parameters.

3. Client to Authorization Server

As specified in the ACE framework section 5.5 [I-D.ietf-ace-oauth-authz], the Client and AS can also use CoAP instead of HTTP to communicate via the token resource. This section specifies how to use OSCORE between Client and AS together with CoAP.

The use of OSCORE for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The client and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the client and the AS communicate using OSCORE ([I-D.ietf-core-object-security]) through the introspection resource as specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

4. Resource Server to Authorization Server

As specified in the ACE framework section 5.6 [I-D.ietf-ace-oauth-authz], the RS and AS can also use CoAP instead of HTTP to communicate via the introspection resource. This section specifies how to use OSCORE between RS and AS. The use of OSCORE for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The RS and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the RS and the AS communicate using OSCORE ([I-D.ietf-core-object-security]) through the introspection resource as specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

5. Security Considerations

TBD.

6. Privacy Considerations

TBD.

7. IANA Considerations

TBD. 'coap_oscore' as profile id. Header parameters 'sid', 'rid', 'kdf' and 'slt' for COSE_Key.

8. References

8.1. Normative 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-08 (work in progress), August 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-07 (work in progress), August 2017.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-05 (work in progress), September 2017.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

8.2. Informative References

- [I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-04 (work in progress), October 2015.
- [I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-05 (work in progress), March 2017.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-07 (work in progress), July 2017.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.

Appendix A. Profile Requirements

This section lists the specifications on this profile based on the requirements on the framework, as requested in Appendix C. of [I-D.ietf-ace-oauth-authz].

- o (Optional) discovery process of how the client finds the right AS for an RS it wants to send a request to: Not specified
- o communication protocol the client and the RS must use: CoAP
- o security protocol the client and RS must use: OSCORE
- o how the client and the RS mutually authenticate: Implicitly by possession of a common OSCORE security context
- o Content-format of the protocol messages: "application/cose+cbor"
- o proof-of-possession protocol(s) and how to select one; which key types (e.g. symmetric/asymmetric) supported: OSCORE algorithms; pre-established symmetric keys
- o profile identifier: coap_oscore
- o (Optional) how the RS talks to the AS for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- o how the client talks to the AS for requesting a token: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- o how/if the /authz-info endpoint is protected: Security protocol above
- o (Optional) other methods of token transport than the /authz-info endpoint: no

Appendix B. Using the pop-key with EDHOC (EDHOC+OSCORE)

EDHOC specifies an authenticated Diffie-Hellman protocol that allows two parties to use CBOR [RFC7049] and COSE in order to establish a shared secret key with perfect forward secrecy. The use of Ephemeral Diffie-Hellman Over COSE (EDHOC) [I-D.selander-ace-cose-ecdhe] in this profile in addition to OSCORE, provides perfect forward secrecy (PFS) and the initial proof-of-possession, which ties the proof-of-possession key to an OSCORE security context.

If EDHOC is used together with OSCORE, and the pop-key (symmetric or asymmetric) is used to authenticate the messages in EDHOC, then the AS MUST provision the following data, in response to the access token request:

- o a symmetric or public key (associated to the RS)
- o a key identifier;

How these parameters are communicated depends on the type of key (asymmetric or symmetric). Moreover, the AS MUST signal the use of OSCORE + EDHOC with the 'profile' parameter set to "coap_oscore_edhoc" and follow Appendix B to derive the security context to run OSCORE.

Note that in the case described in this section, the 'expires_in' parameter, defined in section 4.2.2. of [RFC6749] defines the lifetime in seconds of both the access token and the shared secret. After expiration, C MUST acquire a new access token from the AS, and run EDHOC again, as specified in this section

B.1. Using Asymmetric Keys

In case of an asymmetric key, C MUST communicate its own asymmetric key to the AS in the 'cnf' parameter of the access token request, as specified in section 5.5.1 of [I-D.ietf-ace-oauth-authz]; the AS MUST communicate the RS's public key to C in the response, in the 'rs_cnf' parameter, as specified in section 5.5.1 of [I-D.ietf-ace-oauth-authz]. Note that the RS's public key MUST include a 'kid' parameter, and that the value of the 'kid' MUST be included in the access token, to let the RS know which of its public keys C used. If the access token is a CWT [I-D.ietf-ace-cbor-web-token], the key identifier MUST be placed directly in the 'cnf' structure (if the key is only referenced).

Figure 3 shows an example of such a request in CBOR diagnostic notation without tag and value abbreviations.

```

Header: POST (Code=0.02)
Uri-Host: "server.example.com"
Uri-Path: "token"
Content-Type: "application/cose+cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "cnf" : {
    "COSE_Key" : {
      "kid" : "client_key"
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfHkKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+lrb7btKLv8EX4'
    }
  }
}

```

Figure 3: Example access token request (OSCORE+EDHOC, asymmetric).

Figure 4 shows an example of a corresponding response in CBOR diagnostic notation without tag and value abbreviations.

```

Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
    (contains "kid" : "client_key")',
  "profile" : "coap_oscore_edhoc",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kid" : "server_key"
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'cGJ90UiglWiGahtagnv8usWxHK2PmfHkKwXPS54m0kT',
      "y" : b64'reASjpkttcsz+lrb7btKLv8EX4IBOL+C3BttVivg+lS'
    }
  }
}

```

Figure 4: Example AS response (EDHOC+OSCORE, asymmetric).

B.2. Using Symmetric Keys

In the case of a symmetric key, the AS MUST communicate the key to the client in the 'cnf' parameter of the access token response, as specified in section 5.5.2. of [I-D.ietf-ace-oauth-authz]. AS MUST also select a key identifier, that MUST be included as the 'kid' parameter either directly in the 'cnf' structure, as in figure 4 of [I-D.ietf-ace-oauth-authz], or as the 'kid' parameter of the COSE_Key, as in figure 6 of [I-D.ietf-ace-oauth-authz].

Figure 5 shows an example of the necessary parameters in the AS response to the access token request when EDHOC is used. The example uses CBOR diagnostic notation without tag and value abbreviations.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
    (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore_edhoc",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'5tOS+h42dkw',
      "k" : b64'+aDg2jjU+eIiOFCa9lObw'
    }
  }
}
```

Figure 5: Example AS response (EDHOC+OSCORE, symmetric).

In both cases, the AS MUST also include the same key identifier as 'kid' parameter in the access token metadata. If the access token is a CWT [I-D.ietf-ace-cbor-web-token], the key identifier MUST be placed inside the 'cnf' claim as 'kid' parameter of the COSE_Key or directly in the 'cnf' structure (if the key is only referenced).

Figure 6 shows an example CWT containing the necessary EDHOC+OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'5tOS+h42dkw',
      "k" : b64'+aDg2jjU+eIiOFca9lObw'
    }
  }
}
```

Figure 6: Example CWT with EDHOC+OSCORE, symmetric case.

All other parameters defining OSCORE security context are derived from EDHOC message exchange, including the master secret (see Appendix C.2 of [I-D.selander-ace-cose-ecdhe]).

B.3. Processing

To provide forward secrecy and mutual authentication in the case of pre-shared keys, pre-established raw public keys or with X.509 certificates it is RECOMMENDED to use EDHOC [I-D.selander-ace-cose-ecdhe] to generate the keying material. EDHOC MUST be used as defined in Appendix C of [I-D.selander-ace-cose-ecdhe], with the following additions and modifications.

The first EDHOC message is sent after the access token is posted to the /authz-info resource of the RS as specified in section 5.7.1 of [I-D.ietf-ace-oauth-authz]. Then the EDHOC message_1 is sent and the EDHOC protocol is initiated [I-D.selander-ace-cose-ecdhe]).

Before the RS continues with the EDHOC protocol and responds to this token submission request, additional verifications on the access token are done: the RS SHALL process the access token according to [I-D.ietf-ace-oauth-authz]. If the token is valid then the RS continues processing EDHOC following Appendix C of [I-D.selander-ace-cose-ecdhe], otherwise it discontinues EDHOC and responds with the error code as specified in [I-D.ietf-ace-oauth-authz].

- o In case the EDHOC verification fails, the RS MUST return an error response to the client with code 4.01 (Unauthorized).
- o 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).

- o 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).
- o If all verifications above succeeds, further communication between client and RS is protected with OSCORE, including the RS response to the OSCORE request.

In the case of EDHOC being used with symmetric keys, the protocol in section 5 of [I-D.selander-ace-cose-ecdhe] MUST be used. If the key is asymmetric, the RS MUST also use an asymmetric key for authentication. This key is known to the client through the access token response (see section 5.5.2 of the ACE framework). In this case the protocol in section 4 of [I-D.selander-ace-cose-ecdhe] MUST be used.

Figure 7 illustrates the message exchanges for using OSCORE+EDHOC (step C in figure 1 of [I-D.ietf-ace-oauth-authz]).

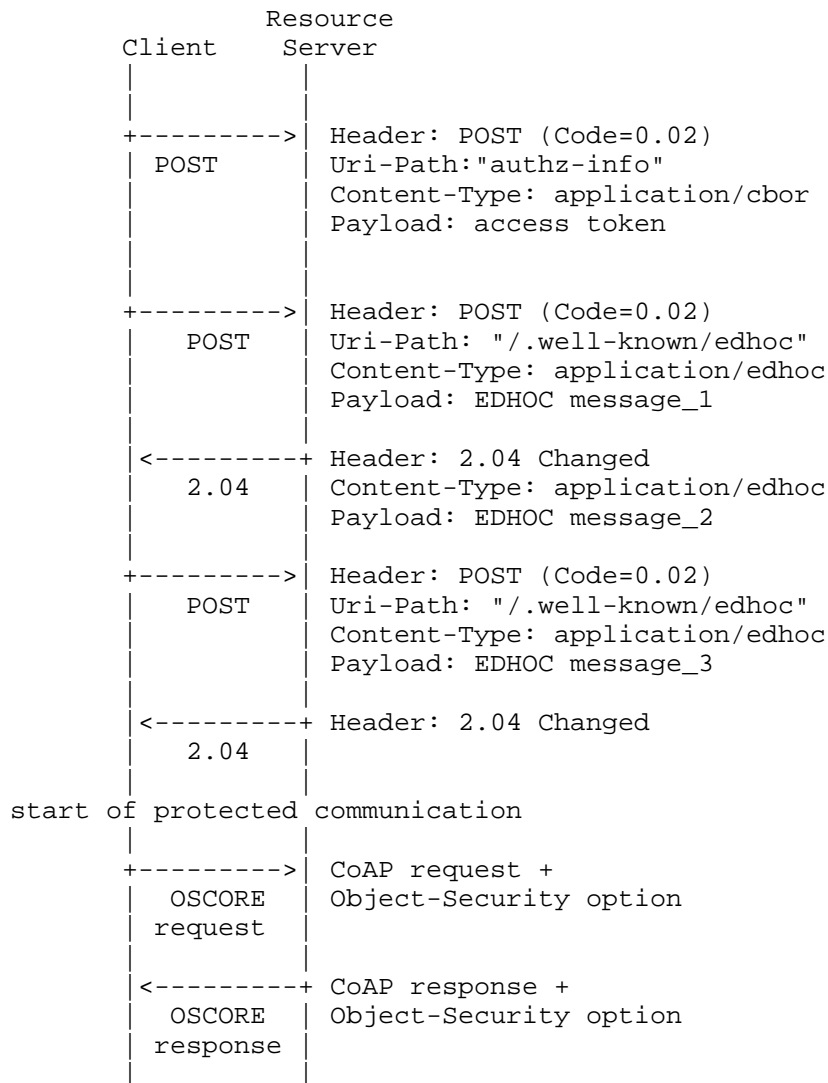


Figure 7: Access token and key establishment with EDHOC

Acknowledgments

The authors wish to thank Jim Schaad, Goeran Selander and Marco Tiloca for the input on this memo. The error responses specified in Appendix B.3 were originally specified by Gerdes et al. in [I-D.gerdes-ace-dcaf-authorize].

Authors' Addresses

Ludwig Seitz
RISE SICS AB
Scheelevagen 17
Lund 22370
SWEDEN

Email: ludwig.seitz@ri.se

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

Email: francesca.palombini@ericsson.com

Martin Gunnarsson
RISE SICS AB
Scheelevagen 17
Lund 22370
SWEDEN

Email: martin.gunnarsson@ri.se

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

M. Tiloca
RISE AB
J. Park
Universitaet Duisburg-Essen
F. Palombini
Ericsson AB
October 22, 2018

Key Management for OSCORE Groups in ACE
draft-tiloca-ace-oscoop-joining-05

Abstract

This document describes a method to request and provision keying material in group communication scenarios where communications are based on CoAP and secured with Object Security for Constrained RESTful Environments (OSCORE). The proposed method delegates the authentication and authorization of new client nodes that join an OSCORE group through a Group Manager server. This approach builds on the ACE framework for Authentication and Authorization, and leverages protocol-specific profiles of ACE to achieve communication security, proof-of-possession and server authentication.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
1.2. Relation to Other Documents	5
2. Protocol Overview	5
2.1. Overview of the Join Process	7
2.2. Overview of the Group Rekeying Process	7
3. Joining Node to Authorization Server	8
3.1. Authorization Request	8
3.2. Authorization Response	9
4. Joining Node to Group Manager	10
4.1. Join Request	10
4.2. Join Response	10
5. Leaving of a Group Member	12
6. Public Keys of Joining Nodes	12
7. Group Rekeying Process	14
8. Security Considerations	14
9. IANA Considerations	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Acknowledgments	17
Authors' Addresses	17

1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] is a method for application-layer protection of the Constrained Application Protocol (CoAP) [RFC7252], using CBOR Object Signing and Encryption (COSE) [RFC8152] and enabling end-to-end security of CoAP payload and options.

As described in [I-D.ietf-core-oscore-groupcomm], OSCORE may be used to protect CoAP group communication over IP multicast [RFC7390]. This relies on a Group Manager, which is responsible for managing an OSCORE group, where members exchange CoAP messages secured with OSCORE. The Group Manager can be responsible for multiple groups, coordinates the join process of new group members, and is entrusted with the distribution and renewal of group keying material.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines a method to:

- o Authorize a node to join an OSCORE group, and provide it with the group keying material to communicate with other group members.
- o Provide updated keying material to group members upon request.
- o Renew the group keying material and distribute it to the OSCORE group (rekeying) upon changes in the group membership.

A client node joins an OSCORE group through a resource server acting as Group Manager for that group. The join process relies on an Access Token, which is bound to a proof-of-possession key and authorizes the client to access a specific join resource at the Group Manager.

Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm] for provisioning and renewing keying material in group communication scenarios.

In order to achieve communication security, proof-of-possession and server authentication, the client and the Group Manager leverage protocol-specific profiles of ACE. These include also possible forthcoming profiles that comply with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Readers are expected to be familiar with the terms and concepts related to the CoAP protocol described in [RFC7252][RFC7390]. Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS and /authz-info at the RS. This

document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

Readers are expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE [I-D.ietf-core-object-security] also in group communication scenarios [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications are secured with OSCORE. In this specification, the Group Manager acts as Resource Server.

This document refers also to the following terminology.

- o **Joining node:** a network node intending to join an OSCORE group, where communication is based on CoAP [RFC7390] and secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].
- o **Join process:** the process through which a joining node becomes a member of an OSCORE group. The join process is enforced and assisted by the Group Manager responsible for that group.
- o **Join resource:** a resource hosted by the Group Manager, associated to an OSCORE group under that Group Manager. A join resource is identifiable with the Group Identifier (Gid) of the respective group. A joining node accesses a join resource to start the join process and become a member of that group.
- o **Join endpoint:** an endpoint at the Group Manager associated to a join resource.
- o **Requester:** member of an OSCORE group that sends request messages to other members of the group.
- o **Listener:** member of an OSCORE group that receives request messages from other members of the group. A listener may reply back, by sending a response message to the requester which has sent the request message.
- o **Pure listener:** member of a group that is configured as listener and never replies back to requesters after receiving request messages. This corresponds to the term "silent server" used in [I-D.ietf-core-oscore-groupcomm].
- o **Group rekeying process:** the process through which the Group Manager renews the security parameters and group keying material, and (re-)distributes them to the OSCORE group members.

1.2. Relation to Other Documents

Figure 1 overviews the main documents related to this specification. Arrows and asterisk-arrows denote normative references and informative references, respectively.

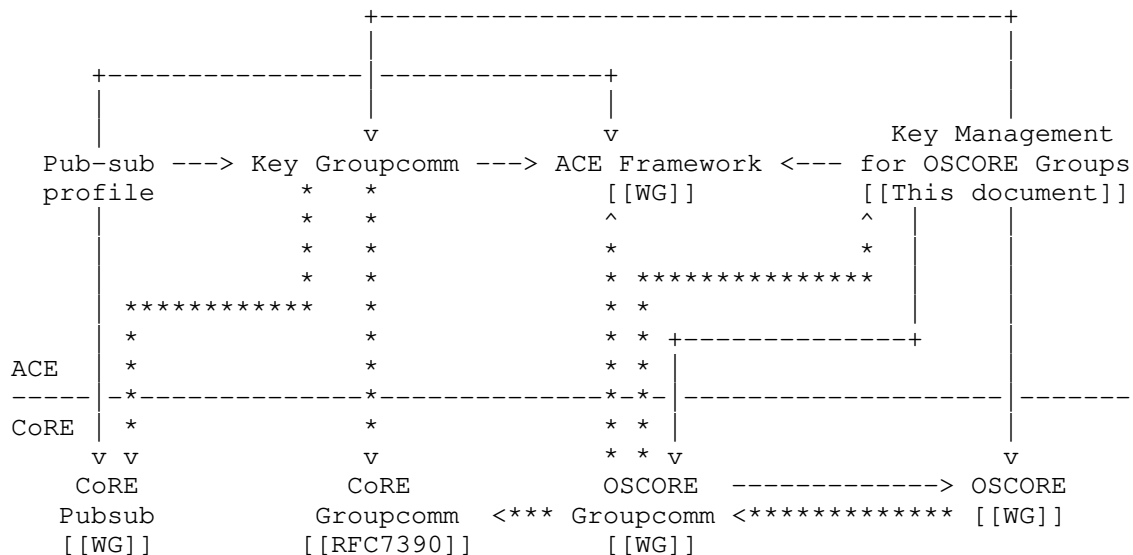


Figure 1: Related Documents

2. Protocol Overview

Group communication for CoAP over IP multicast has been enabled in [RFC7390] and can be secured with Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] as described in [I-D.ietf-core-oscore-groupcomm]. A network node joins an OSCORE group by interacting with the responsible Group Manager. Once registered in the group, the new node can securely exchange messages with other group members.

This specification describes how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to:

- o Enable a node to join an OSCORE group through the Group Manager and receive the security parameters and keying material to communicate with the other members of the group.
- o Enable members of OSCORE groups to retrieve updated group keying material from the Group Manager.

- o Enable the Group Manager to renew the security parameters and group keying material, and to (re-)distribute them to the members of the OSCORE group (rekeying).

With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- o The Group Manager acts as Resource Server (RS), and hosts one join resource for each OSCORE group it manages. Each join resource is exported by a distinct join endpoint. During the join process, the Group Manager provides joining nodes with the parameters and keying material for taking part to secure communications in the OSCORE group. The Group Manager also maintains the group keying material and performs the group rekeying process to distribute updated keying material to the group members.
- o The joining node acts as Client (C), and requests to join an OSCORE group by accessing the related join endpoint at the Group Manager.
- o The Authorization Server (AS) authorizes joining nodes to join OSCORE groups under their respective Group Manager. Multiple Group Managers can be associated to the same AS. The AS MAY release Access Tokens for other purposes than joining OSCORE groups under registered Group Managers. For example, the AS may also release Access Tokens for accessing resources hosted by members of OSCORE groups.

All communications between the involved entities rely on the CoAP protocol and MUST be secured.

In particular, communications between the joining node and the Group Manager leverage protocol-specific profiles of ACE to achieve communication security, proof-of-possession and server authentication. To this end, the AS must signal the specific profile to use, consistently with requirements and assumptions defined in the ACE framework [I-D.ietf-ace-oauth-authz].

With reference to the AS, communications between the joining node and the AS (/token endpoint) as well as between the Group Manager and the AS (/introspect endpoint) can be secured by different means, for instance using DTLS [RFC6347] or OSCORE [I-D.ietf-core-object-security]. Further details on how the AS secures communications (with the joining node and the Group Manager) depend on the specifically used profile of ACE, and are out of the scope of this specification.

2.1. Overview of the Join Process

A node performs the following steps in order to join an OSCORE group. Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm], and are further specified in Section 3 and Section 4 of this document. The Group Manager acts as the Key Distribution Center (KDC) defined in [I-D.palombini-ace-key-groupcomm].

1. The joining node requests an Access Token from the AS, in order to access a join resource on the Group Manager and hence join the associated OSCORE group (see Section 3). The joining node will start or continue using a secure communication channel with the Group Manager, according to the response from the AS.
2. The joining node transfers authentication and authorization information to the Group Manager by posting the obtained Access Token (see Section 4). After that, a joining node must have a secure communication channel established with the Group Manager, before starting to join an OSCORE group under that Group Manager (see Section 4). Possible ways to provide a secure communication channel are DTLS [RFC6347] and OSCORE [I-D.ietf-core-object-security].
3. The joining node starts the join process to become a member of the OSCORE group, by accessing the related join resource hosted by the Group Manager (see Section 4).
4. At the end of the join process, the joining node has received from the Group Manager the parameters and keying material to securely communicate with the other members of the OSCORE group.
5. The joining node and the Group Manager maintain the secure channel, to support possible future communications.

All further communications between the joining node and the Group Manager MUST be secured, for instance with the same secure channel mentioned in step 2.

2.2. Overview of the Group Rekeying Process

If the application requires backward and forward security, the Group Manager MUST generate new security parameters and group keying material, and distribute them to the group (rekeying) upon membership changes.

That is, the group is rekeyed when a node joins the group as a new member, or after a current member leaves the group. By doing so, a

joining node cannot access communications in the group prior its joining, while a leaving node cannot access communications in the group after its leaving.

Parameters and keying material include a new Group Identifier (Gid) for the group and a new Master Secret for the OSCORE Common Security Context of that group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

The Group Manager MUST support the Group Rekeying Process described in Section 7. Future application profiles may define alternative message formats and distribution schemes to perform group rekeying.

3. Joining Node to Authorization Server

This section describes how the joining node interacts with the AS in order to be authorized to join an OSCORE group under a given Group Manager. In particular, it considers a joining node that intends to contact that Group Manager for the first time.

The message exchange between the joining node and the AS consists of the messages Authorization Request and Authorization Response defined in Section 3 of [I-D.palombini-ace-key-groupcomm].

In case the specific AS associated to the Group Manager is unknown to the joining node, the latter can rely on mechanisms like the Unauthorized Resource Request message described in Section 5.1.1 of [I-D.ietf-ace-oauth-authz] to discover the correct AS to contact.

3.1. Authorization Request

The joining node contacts the AS, in order to request an Access Token for accessing the join resource hosted by the Group Manager and associated to the OSCORE group. The Access Token request sent to the /token endpoint follows the format of the Authorization Request message defined in Section 3.1 of [I-D.palombini-ace-key-groupcomm]. In particular:

- o The 'scope' parameter MUST be present and MUST include:
 - * in the first element, either the Group Identifier (Gid) of the group to join under the Group Manager, or a value from which the Group Manager can derive the Gid of the group to join. It is up to the application to define how the Group Manager possibly performs the derivation of the full Gid. Appendix C of [I-D.ietf-core-oscore-groupcomm] provides an example of structured Gid, composed of a fixed part, namely Group Prefix, and a variable part, namely Group Epoch.

- * in the second element, the role(s) that the joining node intends to have in the group it intends to join. Possible values are: "requester"; "listener"; and "pure listener". Possible combinations are: ["requester" , "listener"]; ["requester" , "pure listener"].
- o The 'req_aud' parameter MUST be present and is set to the identifier of the Group Manager.

3.2. Authorization Response

The AS is responsible for authorizing the joining node to join specific OSCORE groups, according to join policies enforced on behalf of the respective Group Manager.

In case of successful authorization, the AS releases an Access Token bound to a proof-of-possession key associated to the joining node.

Then, the AS provides the joining node with the Access Token as part of an Access Token response, which follows the format of the Authorization Response message defined in Section 3.2 of [I-D.palombini-ace-key-groupcomm].

The 'exp' parameter MUST be present. Other means for the AS to specify the lifetime of Access Tokens are out of the scope of this specification.

The AS must include the 'scope' parameter in the response when the value included in the Access Token differs from the one specified by the joining node in the request. In such a case, the second element of 'scope' MUST be present and includes the role(s) that the joining node is actually authorized to take in the group, encoded as specified in Section 3.1 of this document.

Also, the 'profile' parameter indicates the specific profile of ACE to use for securing communications between the joining node and the Group Manager (see Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]).

In particular, if symmetric keys are used, the AS generates a proof-of-possession key, binds it to the Access Token, and provides it to the joining node in the 'cnf' parameter of the Access Token response. Instead, if asymmetric keys are used, the joining node provides its own public key to the AS in the 'req_cnf' parameter of the Access Token request. Then, the AS uses it as proof-of-possession key bound to the Access Token, and provides the joining node with the Group Manager's public key in the 'rs_cnf' parameter of the Access Token response.

4. Joining Node to Group Manager

First, the joining node posts the Access Token to the /authz-info endpoint at the Group Manager, in accordance with the Token post defined in Section 3.3 of [I-D.palombini-ace-key-groupcomm]. Then, the joining node establishes a secure channel with the Group Manager, according to what is specified in the Access Token response and to the signalled profile of ACE.

4.1. Join Request

Once a secure communication channel with the Group Manager has been established, the joining node requests to join the OSCORE group, by accessing the related join resource at the Group Manager.

In particular, the joining node sends to the Group Manager a confirmable CoAP request, using the method POST and targeting the join endpoint associated to that group. This join request follows the format and processing of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm]. In particular:

- o The 'get_pub_keys' parameter is present only if the joining node wants to retrieve the public keys of the group members from the Group Manager during the join process (see Section 6). Otherwise, this parameter MUST NOT be present.
- o The 'client_cred' parameter, if present, includes the public key of the joining node. This parameter MAY be omitted if: i) public keys are used as proof-of-possession keys between the joining node and the Group Manager; or ii) the joining node is asking to access the group exclusively as pure listener; or iii) the Group Manager already acquired this information during a previous join process. In any other case, this parameter MUST be present.

4.2. Join Response

The Group Manager processes the request according to [I-D.ietf-ace-oauth-authz]. If this yields a positive outcome, the Group Manager updates the group membership by registering the joining node as a new member of the OSCORE group.

The Group Manager replies to the joining node providing the updated security parameters and keying material necessary to participate in the group communication. This join response follows the format and processing of the Key Distribution success Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm]. In particular:

- o The 'key' parameter includes what the joining node needs in order to set up the OSCORE Security Context as per Section 2 of [I-D.ietf-core-oscore-groupcomm]. In particular:
 - * The 'kty' parameter has value "Symmetric".
 - * The 'k' parameter includes the OSCORE Master Secret.
 - * The 'exp' parameter specifies when the OSCORE Security Context derived from these parameters expires.
 - * The 'alg' parameter, if present, has as value the AEAD algorithm used in the group.
 - * The 'kid' parameter, if present, has as value the identifier of the key in the parameter 'k'.
 - * The 'base IV' parameter, if present, has as value the OSCORE Common IV.
 - * The 'clientID' parameter, if present, has as value the OSCORE Sender ID assigned to the joining node by the Group Manager. This parameter is not present if the node joins the group exclusively as pure listener, according to what specified in the Access Token (see Section 3.2). In any other case, this parameter MUST be present.
 - * The 'serverID' parameter MUST be present and has as value the Group Identifier (Gid) associated to the group.
 - * The 'kdf' parameter, if present, has as value the KDF algorithm used in the group.
 - * The 'slt' parameter, if present, has as value the OSCORE Master Salt.
 - * The 'cs_alg' parameter MUST be present and has as value the countersignature algorithm used in the group.
- o The 'pub_keys' parameter is present only if the 'get_pub_keys' parameter was present in the join request. If present, this parameter includes the public keys of the group members that are relevant to the joining node. That is, it includes: i) the public keys of the non-pure listeners currently in the group, in case the joining node is configured (also) as requester; and ii) the public keys of the requesters currently in the group, in case the joining node is configured (also) as listener or pure listener.

- o The 'group_policies' parameter SHOULD be present and includes a list of parameters indicating particular policies enforced in the group. For instance, it can indicate the method to achieve synchronization of sequence numbers among group members (see Appendix E of [I-D.ietf-core-oscore-groupcomm]).

Finally, the joining node uses the information received in the join response to set up the OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. From then on, the joining node can exchange group messages secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].

If the application requires backward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to all the current group members (see Section 7).

When the OSCORE Master Secret expires, as specified by 'exp' in the 'key' parameter of the join response, the node considers the OSCORE Security Context also invalid and to be renewed. Then, the node retrieves updated security parameters and keying material, by exchanging shortened Join Request and Join Response messages with the Group Manager, according to the approach defined in Section 6 of [I-D.palombini-ace-key-groupcomm]. Finally, the node uses the updated security parameters and keying material to set up the new OSCORE Security Context as described in Section 2 of [I-D.ietf-core-oscore-groupcomm].

5. Leaving of a Group Member

A node may be removed from the OSCORE group, due to expired or revoked authorization, or after its own request to the Group Manager.

If the application requires forward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to the remaining group members (see Section 7). The leaving node must not be able to acquire the new security parameters and group keying material distributed after its leaving.

Same considerations in Section 5 of [I-D.palombini-ace-key-groupcomm] apply here as well, considering the Group Manager acting as KDC. In particular, a node requests to leave the OSCORE group as described in Section 5.2 of [I-D.palombini-ace-key-groupcomm].

6. Public Keys of Joining Nodes

Source authentication of OSCORE messages exchanged within the group is ensured by means of digital counter signatures (see Sections 2 and 3 of [I-D.ietf-core-oscore-groupcomm]). Therefore, group members

must be able to retrieve each other's public key from a trusted key repository, in order to verify source authenticity of incoming group messages.

As also discussed in [I-D.ietf-core-oscore-groupcomm], the Group Manager acts as trusted repository of the public keys of the group members, and provides those public keys to group members if requested to. Upon joining an OSCORE group, a joining node is thus expected to provide its own public key to the Group Manager.

In particular, four cases can occur when a new node joins a group.

- o The joining node is going to join the group exclusively as pure listener. That is, it is not going to send messages to the group, and hence to produce signatures with its own private key. In this case, the joining node is not required to provide its own public key to the Group Manager upon joining the group.
- o The Group Manager already acquired the public key of the joining node during a previous join process. In this case, the joining node may not provide again its own public key to the Group Manager, in order to limit the size of the join request.
- o The joining node and the Group Manager use an asymmetric proof-of-possession key to establish a secure communication channel. In this case, the Group Manager stores the proof-of-possession key conveyed in the Access Token as the public key of the joining node.
- o The joining node and the Group Manager use a symmetric proof-of-possession key to establish a secure communication channel. In this case, upon performing a join process with that Group Manager for the first time, the joining node specifies its own public key in the 'client_cred' parameter of the join request targeting the join endpoint (see Section 4.1).

Furthermore, as described in Section 4.1, the joining node may have explicitly requested the Group Manager to retrieve the public keys of the current group members, i.e. through the 'get_pub_keys' parameter in the join request. In this case, the Group Manager includes also such public keys in the 'pub_keys' parameter of the join response (see Section 4.2).

Later on as a group member, the node may need to retrieve the public keys of other group members. The node can do that by exchanging shortened Join Request and Join Response messages with the Group Manager, according to the approach defined in Section 7 of [I-D.palombini-ace-key-groupcomm].

7. Group Rekeying Process

In order to rekey the OSCORE group, the Group Manager distributes a new Group ID of the group and a new OSCORE Master Secret for that group. To this end, the Group Manager MUST support at least the following group rekeying scheme. Future application profiles may define alternative message formats and distribution schemes.

The Group Manager uses the same format of the Join Response message in Section 4.2. In particular:

- o Only the 'key' parameter is present.
- o The 'k' parameter of the 'key' parameter includes the new OSCORE Master Secret.
- o The 'serverID' parameter of the 'key' parameter includes the new Group ID.

The Group Manager separately sends a group rekeying message to each group member to be rekeyed. Each rekeying message MUST be secured with the pairwise secure communication channel between the Group Manager and the group member used during the join process.

8. Security Considerations

The method described in this document leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

- o Management of group keying material (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager is responsible for the renewal and re-distribution of the keying material in the groups of its competence (rekeying). According to the specific application requirements, this can include rekeying the group upon changes in its membership. In particular, renewing the keying material is required upon a new node's joining or a current node's leaving, in case backward security and forward security have to be preserved, respectively.
- o Provisioning and retrieval of public keys (see Section 2 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager acts as key repository of public keys of group members, and provides them upon request.
- o Synchronization of sequence numbers (see Section 5 of [I-D.ietf-core-oscore-groupcomm]). This concerns how a listener

node that has just joined an OSCORE group can synchronize with the sequence number of requesters in the same group.

Before sending the join response, the Group Manager should verify that the joining node actually owns the associated private key, for instance by performing a proof-of-possession challenge-response, whose details are out of the scope of this specification.

Further security considerations are inherited from [I-D.palombini-ace-key-groupcomm], the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the specific profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

9. IANA Considerations

This document has no actions for IANA.

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) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-16 (work in progress), October 2018.

[I-D.ietf-ace-oscore-profile]

Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-04 (work in progress), October 2018.

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

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

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-02 (work in progress), June 2018.

- [I-D.palombini-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-palombini-ace-key-groupcomm-02 (work in progress), October 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.ietf-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-ietf-ace-dtls-authorize-05 (work in progress), October 2018.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Martin Gunnarsson, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok for their comments and feedback.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-164 29 Stockholm
Sweden

Email: marco.tiloca@ri.se

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

ace
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

H. Tschofenig
ARM Ltd.
W. Werner
Werner Management Services e.U.
October 30, 2017

Security for Low-Latency Group Communication
draft-tschofenig-ace-group-communication-security-00.txt

Abstract

Some Internet of Things application domains require secure group communication. This draft describes procedures for authorization, key management, and securing group messages. We specify the usage of object security at the application layer for group communication and assume that CoAP is used as the application layer protocol. The architecture allows the usage of symmetric and asymmetric keys to secure the group messages. The asymmetric key solution provides the ability to uniquely authenticate the source of all group messages and this is the recommended architecture for most applications. However, some applications have strict requirements on latency for group communication (e.g. in non-emergency lighting applications) and it may not always be feasible to use the secure source authenticated architecture. In such applications we recommend the use of dynamically generated symmetric group keys to secure group communications.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	3
3. Architecture - Group Authentication	5
3.1. Assumptions	8
3.2. AT-KDC Access Tokens	9
3.3. AT-R Access Tokens	9
3.4. Multicast Message Content	10
3.5. Receiver Algorithm	11
3.6. Sender Algorithm	12
4. Architecture - source authentication	14
4.1. Assumptions	16
4.2. AT-R Access Tokens	17
4.3. Multicast Message Content	17
4.4. Receiver Algorithm	18
4.5. Sender Algorithm	19
5. Security Considerations	20
5.1. Applicability statement	20
5.2. Token Verification	21
5.3. Token Revocation	21
5.4. Time	22
6. Operational Considerations	22
6.1. Persistence of State Information	22
6.2. Provisioning in Small Networks	23
6.3. Client IDs	23
6.4. Application Groups vs. Security Groups	23
6.5. Lost/Stolen Device	23
7. Acknowledgements	24
8. IANA Considerations	24
9. Contributors	24
10. References	24
10.1. Normative References	24

10.2. Informative References	25
Appendix A. Access Levels	25
Authors' Addresses	26

1. Introduction

There are low latency group communication use cases that require securing communication between a sender, or a group of senders, and a group of receivers. In the lighting use case, a set of lighting nodes (e.g., luminaires, wall-switches, sensors) are grouped together into a single "Application Group" and the following three requirements need to be addressed:

1. Only authorized members of the application group must be able to read and process messages.
2. Receivers of group messages must be able to verify the integrity of received messages as being generated within the group.
3. Message communication and processing must happen with a low latency and in synchronous manner.

This document discusses a group communication security solution that satisfies these three requirements. As discussed in Section 4, we recommend the usage of an asymmetric key solution that allows unique source authentication of all group messages. However, in situations where the low latency requirements can not be met (e.g. in non-emergency lighting applications), the alternative architecture discussed in Section 3 based on symmetric keys is recommended.

2. Terminology

This document uses the following terms from [I-D.ietf-ace-actors]: Authorization Server, Resource Owner, Client, Resource Server. The terms 'sender' and 'receiver' refer to the application layer messaging used for lighting control; other communication interactions with the supporting infrastructure uses unicast messaging.

When nodes are combined into groups there are different layers of those groups with unique characteristics. For clarity we introduce terminology for three different groups:

Application Group:

An application group consists of the set of all nodes that have been configured to respond to a single application layer request. For example, a wall mounted switch and a set of luminaires in a single room might belong to a single group and the switch may be

used to turn on/off all the luminaires in the group simultaneously with a single button press. In the remainder of this document we will use GID to identify an application group.

Multicast Group:

A multicast group consists of the set of all nodes that subscribe to the same multicast IP address.

Security Group:

A security group consists of the set of all nodes that have been provisioned with the same keying material. All the nodes within a security group share a security association or a sequence of security associations wherein a single association specifies the keying material, algorithm-specific information, lifetime and a key ID.

Source-authenticated Security Group:

A source-authenticated security group consists of the set of receiver nodes that have been provisioned with the public verification keying material of all the sender nodes and the set of sender nodes that are provisioned with their unique private signing keying material. All the nodes within a source-authenticated security group share a security association or a sequence of security associations wherein a single association specifies the the public or private keying material, algorithm-specific information, lifetime and a key ID.

Typically, the four groups might not coincide due to the memory constraints on the devices and also security considerations. For instance, in a small room with windows, we may have three application groups: "room group", "luminaires close to the window group" and "luminaires far from the window group". However, we may choose to use only one multicast group for all devices in the room and one security group for all the devices in the room. Note that every application group belongs to a unique security group. However, the converse is not always true. This implies that the application group ID maybe used to determine the associated security group but not vice versa.

The fact that security groups may not coincide with application groups implies that

- (1) an application must be able to specify which resources on a resource server are accessible by a client that has access to the group key, and

(2) a method is required to associate the group key to the application group(s) for which the group key may be used.

In this document we provide fields that may be used to specify the "scope of the key" and "application groups for which the key may be used". A commissioner has a lot of flexibility to assign nodes to multicast groups and to security groups while the application groups will be determined by the semantics of the application itself. The exact partitioning of the nodes into security and multicast groups is therefore deployment specific.

3. Architecture - Group Authentication

Each node in a lighting application group might be a sender, a receiver or both sender and receiver (even though in Figure 1, we show nodes that are only senders or only receivers for clarity). The low latency requirement implies that most of the communication between senders and receivers of application layer messages is done using multicast IP. On some occasions, a sender in a group will be required to send unicast messages to unique receivers within the same group and these unicast messages also need communication security.

Two logical entities are introduced and they have the following function:

Key Distribution Center (KDC): This logical entity is responsible for generating symmetric keys and distributing them to the nodes authorized to receive them. The KDC ensures that nodes belonging to the same security group receive the same key and that the keys are renewed based on certain events, such as key expiry or change in group membership.

Authorization Server (AS): This logical entity stores authorization information about devices, meta-data about them, and their roles in the network. For example, a luminaire is associated with different groups, and may have meta-data about its location in a building.

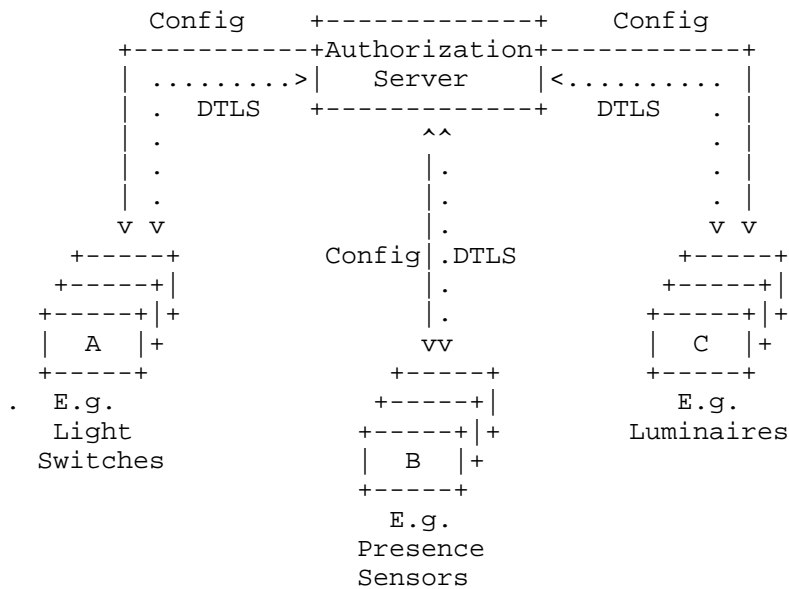
Note that we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 1 and Figure 2 provide an architectural overview. The dotted lines illustrate the use of unicast DTLS messages for securing the message exchange between all involved parties. The secured group messages between senders and receivers are indicated using lines with

star/asterisk characters. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) which are to be defined.

Figure 1 illustrates the information flow between an authorization server and the nodes participating in the lighting network, which includes all nodes that exchange lighting application messages. This step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control lights, may require an authorization server that dynamically executes access control decisions.

Figure 1 shows the commissioning phase where the nodes obtain configuration information, which includes the AT-KDC. The AT-KDC is an access token and includes authorization claims for consumption by the key distribution center. We use the access token terminology from [RFC6749]. The AT-KDC in this architecture may be a bearer token or a proof-of-possession (PoP) token. The bearer token concept is described in [RFC6750] and the PoP token concept is explained in [I-D.ietf-oauth-pop-architecture]. The AT-KDC is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-KDC is protected against modifications using a digital signature or a message authentication code. It is verified in Figure 2 by the KDC.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-KDC) and other meta-data.

Figure 1: Architecture: Commissioning Phase.

In the simplified message exchange shown in Figure 2 a sender requests a security group key and the access token for use with the receivers (called AT-R). The request contains information about the resource it wants to access, such as the application group and other resource-specific information, if applicable, and the previously obtained AT-KDC access token. Once the sender has successfully obtained the requested information it starts communicating with receivers in that group using group messages. The symmetric key obtained from the KDC is used to secure the groups messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary group key to verify the incoming messages and to determine what resource the requestor is authorized to access. Both pieces of information can be found in the AT-R access token.

Group messages need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using

a keyed message digest in combination with the group key. The use of symmetric keys is envisioned in this specification due to latency requirements. For unicast messaging between the group members and the AS or KDC, we assume the use of DTLS for transport security. However, the use of TLS, and application layer security is possible but is outside the scope of this document.

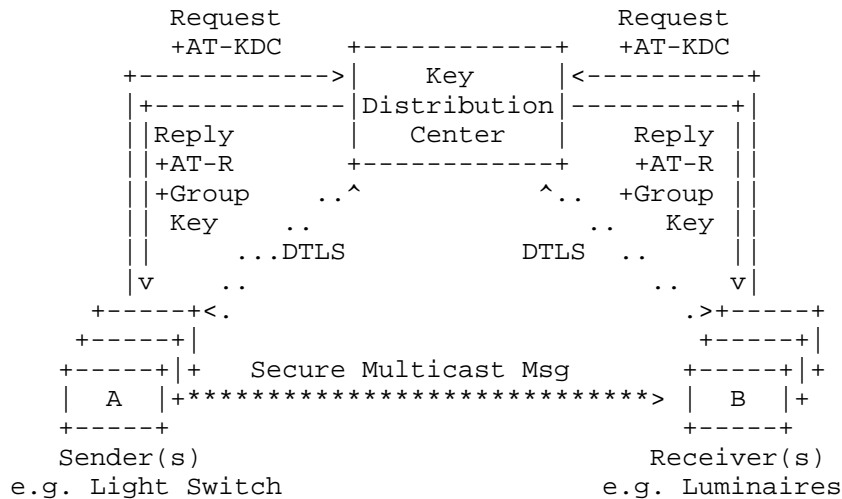


Figure 2: Architecture: Group Key Distribution Phase.

3.1. Assumptions

1. The AT-KDC is a manifestation of the authorization granted to a specific client (or user running a client). The AT-KDC is longer-lived and can be used to request multiple AT-Rs.
2. Each AT-R is valid for use with one or multiple application groups.
3. The AS and the KDC logical roles may reside in different physical entities.
4. The AT-KDC as well as the AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.
5. The AT-KDC token is opaque to the client. Data that is meant for processing by the client has to be conveyed to the client

separately. The AT-R token on the other hand is meant for consumption by the client.

6. The client requests AT-Rs for different application groups by including additional information in the request to the KDC for what application groups the AT-R(s) have to be requested. The KDC may return multiple AT-Rs in a single response (for performance reasons).
7. The AT-KDC and the AT-R are encoded as CBOR Web Tokens [I-D.ietf-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msgl].

3.2. AT-KDC Access Tokens

The AT-KDC contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Recipient/Audience: Indication to whom the AT-KDC was issued to. In this case, it is the KDC.
5. Client ID: Information about the client that was authenticated by the authorization server.
6. Issued at: Indicates date and time when the AT-KDC was created by the authorization server.

3.3. AT-R Access Tokens

Clients send the AT-KDC to the KDC in order to receive an AT-R.

The KDC MUST maintain a table consisting of scope values, which includes the application group id. These entries point to a sequence of security associations. A security association specifies the key material, algorithm-specific information, lifetime and a key ID and the key ID may be used to identify this security association.

The AS/KDC must guarantee the uniqueness of the client ids for its nodes. This may be accomplished by the AS/KDC assigning values to the nodes or by using information that is already unique per device (such as an EUI-64).

The KDC furthermore needs to be configured with information about the authorization servers it trusts. This may include a provisioned trust anchor store, or shared credentials (similar to a white list).

The KDC MUST generate new group keys after the validity period of the current group key expires.

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Security Group Key: Key to use for the group communication.
5. Algorithm: Used for secure group communication.
6. KID: Sequentially increasing ID of the key for the security group (the devices may store an older key to help with key rolling.)
7. Issued at: Indicates date and time when the AT-R was created by the KDC.

3.4. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of
 - * Client ID (unencrypted, integrity protected): Every sender managed by a key distribution center MUST have a unique client ID.

- * Sequence Number (unencrypted, integrity protected): Used for replay protection.
- * An implicit IV that is either derived from the keys at the end-points or fixed to a certain value by standard (not sent in the message)

2. MAC (not integrity protected): For integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.
2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.
3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.
4. Key ID (unencrypted, integrity protected): Is used to select the current security association from the sequence of security associations identified by the application group id.
5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)
6. CoAP Payload (encrypted, integrity protected).

3.5. Receiver Algorithm

All receiving devices MUST maintain a table consisting of mappings of application group id, to a sequence of security associations.

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association.

The key id is used for situations where the group key is updated by the KDC (for example in situations where a device in a group is lost or stolen).

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver **MUST** silently discard an incoming message in the following cases:

- o Application Group ID lookup does not return any security association.
- o Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- o Integrity check fails.
- o Decryption fails.
- o Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association **MUST** be updated to the nonce value in the message.

3.6. Sender Algorithm

Figure 3 describes the algorithm for obtaining the necessary credentials to transmit a secure group message. When the sender wants to send a message to the application group, it checks if it has the respective group key. If no group key is available then it determines whether it has an access token for use with the KDC (i.e., AT-KDC). If no AT-KDC is found in the cache then it contacts the authorization server to obtain that AT-KDC. Note that this assumes

that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported. In the other case where the AT-KDC is already available the sender contacts the KDC to obtain a group key. If a group key is already available then the sender can transmit a secured message to the group immediately.

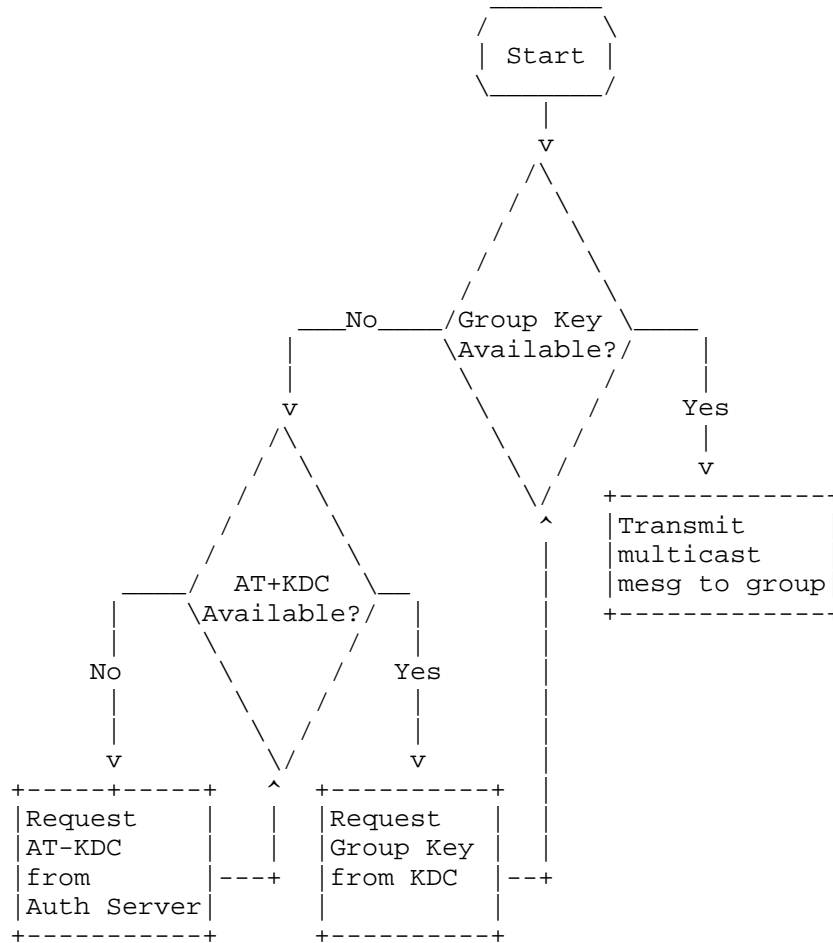


Figure 3: Steps to Transmit Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a group key; the sender is likely to be

pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

Group keys have a lifetime, which is configuration-dependent, but mechanisms need to be provided to update the group keys either via the sender asking for a group key renewal or via the KDC pushing new keys to senders and receivers. The lifetime can be based on time or on the number of transmitted messages.

4. Architecture - source authentication

This section discusses the usage of asymmetric keys to achieve source authentication of group messages and is the recommended architecture for securing group messages. However, this solution may not meet the low latency requirement without adequate hardware support but still most of the group communication between senders and receivers of application layer messages is done using multicast IP.

Unlike the previous architecture, the current architecture requires only the Authorization Server (AS) logical entity as defined in the previous section.

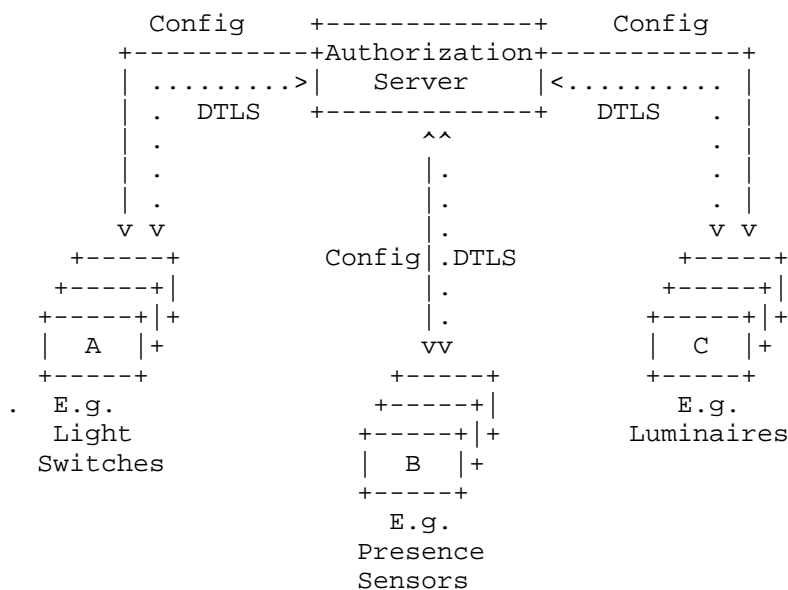
As in the previous case we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 4 and Figure 5 provide an architectural overview for the source authenticated case. The main differences from the previous case is that the AS provides directly the AT-R tokens. Further no KDC is required in this case since the senders and receivers can use their public-private key pair credentials to secure messages. The AS may provide authorization based on the pre-existing device credentials or issue new credentials to the devices. The security of the group messages is accomplished at the application level using small modification to OSCoAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) but based on public key signatures which are to be defined.

Figure 4 illustrates the information flow between an authorization server and the nodes participating in the source-authenticated group network. Like the previous case, this step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control

lights, may require an authorization server that dynamically executes access control decisions.

Figure 4 shows the commissioning phase where the nodes obtain configuration information, which includes directly the AT-R. The AT-R is an access token and includes authorization claims for consumption by the receivers. The AT-R may be a bearer token or a proof-of-possession (PoP) token. The AT-R is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-R is protected against modifications using a digital signature. It is verified in Figure 5 by the receivers.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-R) and other meta-data.

Figure 4: Architecture - Source-authenticated: Commissioning Phase.

In the simplified message exchange shown in Figure 5 a sender starts communicating with receivers in that source-authenticated group using public-key signed group messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary public verification key of the senders (or a root verification key if they are certified by the same authority) to verify the incoming messages and the public verification key of the AS to determine what resource the requestor is authorized to access. Both pieces of information can either be found in the AT-R access token or separately configured during the commissioning phase.

Source-authenticated Group messages also need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using a public-key signature. This may not achieve the latency requirements and used where source-authentication is more important. For unicast messaging between the group members and the AS, we assume the use of DTLS for transport security.

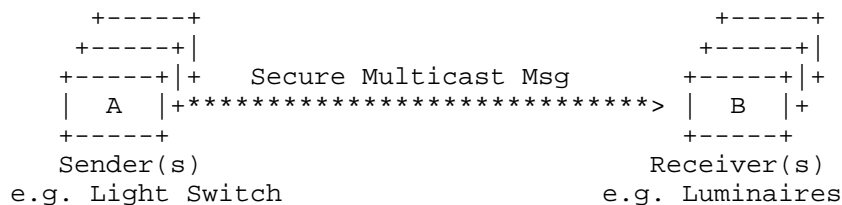


Figure 5: Architecture - Source-authenticated: Group communication.

4.1. Assumptions

1. The AT-R is a manifestation of the authorization granted to a specific client (or user running a client). The AT-R is longer-lived and can be used directly for source-authenticated group communication until it is revoked or expired.
2. Each AT-R is valid for use with one or multiple application groups.
3. The AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.
4. The AT-R token is not opaque to the client and is meant for consumption by the client.
5. The client requests AT-Rs for different application groups by including additional information in the request to the AS for what application groups the AT-R(s) have to be requested. The AS

may return multiple AT-Rs in a single response (for performance reasons).

6. The AT-R is encoded as CBOR Web Tokens [I-D.ietf-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

4.2. AT-R Access Tokens

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).
2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).
3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.
4. Recipient/Audience: Indication to whom the AT-R was issued to. In this case, it is the receivers.
5. Client ID: Information about the client that was authenticated by the authorization server.
6. Client public key: The public key to use for signing the source-authenticated group communication. These public key may be optionally certified using the AS key or a domain root key. This reduces the need for additional per-device public key storage on the receivers.
7. Algorithm: Used for source-authenticated secure group communication.
8. Issued at: Indicates date and time when the AT-R was created by the authorization server.

4.3. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

- * Client ID (unencrypted, integrity protected): Every sender managed by the AS MUST have a unique client ID.
 - * Sequence Number (unencrypted, integrity protected): Used for replay protection.
2. Signature (not integrity protected): For source-authenticated integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.
2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.
3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.
4. Key ID (unencrypted, integrity protected): Is used to select the correct security association containing the verification key from the sequence of security associations identified by the application group id.
5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)
6. CoAP Payload (encrypted, integrity protected).

4.4. Receiver Algorithm

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association to use to verify the message.

The key id is used for situations where the client may have different keys for different applications.

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the

request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- o Application Group ID lookup does not return any security association.
- o Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- o Integrity check fails.
- o Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

4.5. Sender Algorithm

Figure 6 describes the algorithm for obtaining the necessary credentials to transmit a source-authenticated secure group message. When the sender wants to send a message to the application group, it checks if it has the respective signing key that matches the KID in the AT-R. If no signing key is available then it contacts the authorization server to obtain the AT-R and corresponding signing keys. Note that this assumes that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported.

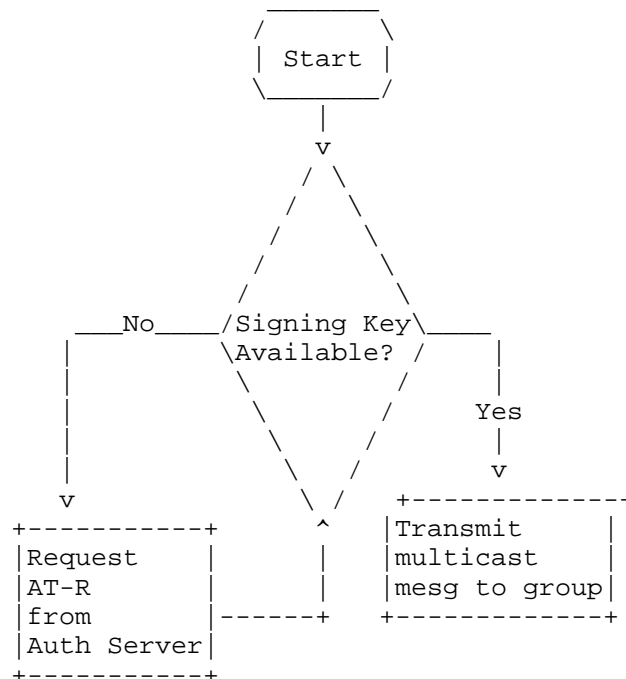


Figure 6: Steps to Transmit Source-authenticated Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a AT-R; the sender is likely to be pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

5. Security Considerations

5.1. Applicability statement

This document describes two architectures based on symmetric group keys in Section 3 and asymmetric keys in Section 4.

The symmetric key solution is based on a group key that is shared between all group members including senders and receivers. As all members of the group possess the same key, it is only possible to authenticate group membership for the source of a message. In particular, it is not possible to authenticate the unique source of a message and consequently it is not possible to authorize a single

node to control a group. Moreover, because the group key is shared across multiple nodes, it may be easier for an attacker to determine the group key by attacking any member of the group (note that this group key is dynamically generated and is usually stored in volatile memory which offers some additional protection). Subsequent to such an attack, it is also difficult to determine which of the group members was compromised and this makes it difficult to return the system to normal operation after an attack.

The asymmetric key solution distinguishes between a sender in the group and the receivers. In particular, the sender is in possession of a private key and the receivers are in possession of the corresponding public key. This allows the unique source of any group message to be authenticated. Moreover, an attacker cannot compromise the system by breaking into any of the receiving nodes. However, for constrained devices, the asymmetric key solution comes at a processing cost with cryptographic computations taking too long.

Therefore, it is recommended that whenever possible, the architecture with source authentication SHOULD be used to secure all multicast communication. However, in less sensitive applications (e.g. controlling luminaires in non-emergency applications), the architecture without source authentication MAY be used. When using the symmetric key solution two mitigating factors could improve system security. It is possible to achieve source authentication of messages at lower layers by requiring unique MAC layer keys for all devices within the network. The symmetric group keys are dynamically generated and therefore SHOULD be stored in volatile memory.

5.2. Token Verification

Due to the low latency requirements, token verification needs to be done locally and cannot be outsourced to other parties. For this reason a self-contained token must be used and the receivers are required to follow the steps outlined in Section 7.2 of RFC 7519 [RFC7519]. This includes the verification of the message authentication code protecting the contents of the token and the encryption envelope protecting the contained symmetric group key.

5.3. Token Revocation

Tokens have a specific lifetime. Setting the lifetime is a policy decision that involves making a trade-off decision. Allowing a longer lifetime increases the need to introduce a mechanism for token revocation (e.g., a real-time signal from the KDC/Authorization Server to the receivers to blacklist tokens) but lowers the communication overhead during normal operation since new tokens need to be obtained only from time to time. Real-time communication with

the receivers to revoke tokens may not be possible in all cases either, particularly when off-line operation is demanded or in small networks where the AS or even the KDC is only present during commissioning time.

We therefore recommend to issue short-lived tokens for dynamic scenarios like users accessing the lighting infrastructure of buildings using smartphones, tablets and alike to avoid potential security problems when tokens are leaked or where authorization rights are revoked. For senders that are statically mounted (like traditional light switches) we recommend a longer lifetime since re-configurations and token leakage is less likely to happen frequently.

To limit the authorization rights, tokens should contain an audience restriction, scoping their use to the intended receivers and to their access level.

5.4. Time

Senders and receivers are not assumed to be equipped with real-time clocks but these devices are still assumed to interact with a time server. The lack of accurate clocks is likely to lead to clock drifts and limited ability to check for replays. For those cases where no time server is available, such as in small network installations, token verification cannot check for expired tokens and hence it might be necessary to fall-back to tokens that do not expire.

6. Operational Considerations

6.1. Persistence of State Information

Devices in the lighting system can often be powered down intentionally or unintentionally. Therefore the devices may need to store the authorization tokens and cryptographic keys (along with replay context) in persistent storage like flash. This is especially required if the authorization server is no more online because it was removed after the commissioning phase. However the decision on the data to be persistently stored is a trade-off between how soon the devices can be back online to normal operational mode and the memory wear caused due to limited program-erase cycles of flash over the 15-20 years life-time of the device.

The different data that may need to be stored are access tokens AT-KDC, AT-R and last seen replay counter.

6.2. Provisioning in Small Networks

In small networks the authorization server and the KDC may be available only temporarily during the commissioning process and are not available afterwards.

6.3. Client IDs

A single device should not be managed by multiple KDCs. However, a group of devices in a domain (such as a lighting installation within an enterprise) should either be managed by a single KDC or, if there are multiple KDCs serving the devices in a given domain, these KDCs MUST exchange information so that the assigned client id and application group id values are unique within the devices in that domain. We assume that only devices within a given domain communicate with each other using group messages.

6.4. Application Groups vs. Security Groups

Multiple application groups may use the same key for performance reasons, reducing the number of keys needed to be stored - leading to less RAM needed by each node. This is only a reasonable option if the attack surface is not increased. For example, a room A is configured to use three application groups to address a subset of the device. In addition to configuring all nodes in room A with these three application groups the nodes are configured with a special group that allows them to access all devices in room A, referred as the all-nodes-in-room-A group. In this case, having the nodes to use the same key for the all-nodes-in-room group and the three groups does not increase the attack surface since any node can already use the all-nodes-in-room-A group to control other devices in that room. The three application groups in room A are a subset of the larger all-nodes-in-room-A group.

6.5. Lost/Stolen Device

The following procedure MUST be implemented if a device is stolen or keys are lost.

1. The AS tells the KDC to invalidate the AT-KDC.
2. The KDC no longer returns a new group key if the invalidated AT-KDC is presented to it.
3. The KDC generates new keys for all security groups to which the compromised device belongs.

The KDC SHOULD inform all devices in the security group to update their group key. This requires the KDC to maintain a list of all devices that belong to the security group and to be able to contact them reliably.

7. Acknowledgements

The author would like to thank Esko Dijk for his help with this document.

Parts of this document are a byproduct of the OpenAIS project, partially funded by the Horizon 2020 programme of the European Commission. It is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of fitness for a particular purpose. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the OpenAIS project or the European Commission.

8. IANA Considerations

This document defines one CoAP Header Option Application Group ID that MUST be allocated in the Registry "CoAP Option Numbers" of [RFC6749]. IANA is requested to allocation TBD option number to application group ID in this specification.

9. Contributors

We would like to thank our former co-authors, Abhinav Somaraju and Sandeep Kumar for their contributions to earlier versions of the draft.

10. References

10.1. 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-cbor-web-token]

Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-09 (work in progress), October 2017.

- [I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)",
draft-ietf-cose-msg-24 (work in progress), November 2016.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.

10.2. Informative References

- [I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H.
Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security
Architecture", draft-ietf-oauth-pop-architecture-08 (work
in progress), July 2016.
- [I-D.selander-ace-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security of CoAP (OSCOAP)", draft-selander-ace-
object-security-06 (work in progress), October 2016.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework",
RFC 6749, DOI 10.17487/RFC6749, October 2012,
<<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
Framework: Bearer Token Usage", RFC 6750,
DOI 10.17487/RFC6750, October 2012,
<<https://www.rfc-editor.org/info/rfc6750>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
(JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,
<<https://www.rfc-editor.org/info/rfc7519>>.

Appendix A. Access Levels

A characteristic of the lighting domain is that access control decisions are also impacted by the type of operation being performed and those categories are listed below. The following access levels are pre-defined.

Level 0: Service detection only

This is a service that is used with broadcast service detection methods. No operational data is accessible at this level.

Level 1: Reporting only

This level allows access to sensor and other (relatively uncritical) operational data and the device error status. The operation of the system cannot be influenced using this level.

Level 2: Standard use

This level allows access to all operational features, including access to operational parameters. This is the highest level of access that can be obtained using (secure) multicast.

Level 3: Commissioning use / Parametrization Services

This level gives access to certain parameters that change the day-to-day operation of the system, but does not allow structural changes.

Level 4: Commissioning use / Localization and Addressing Services

(including Factory Reset) This level allows access to all services and parameters including structural settings.

Level 5: Software Update and related Services

This level allows the change and upgrade of the software of the devices.

Note: The use of group security is disallowed for level higher than Level 2 and unicast communication is used instead.

Authors' Addresses

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

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

Walter Werner
Werner Management Services e.U.
Josef-Anton-Herrburgerstr. 10
Dornbirn 6850
Austria

Email: werner@werner-ms.at

ACE
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2018

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
S. Kumar
Philips Lighting Research
M. Richardson
SSW
M. Furuhed
Nexus Group
S. Raza
RISE SICS
January 22, 2018

EST over secure CoAP (EST-coaps)
draft-vanderstok-ace-coap-est-04

Abstract

Enrollment over Secure Transport (EST) [RFC7030] is used as a certificate management protocol over HTTPS.

Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) [RFC7252] for message exchanges. This document defines how to transport EST payloads over secure CoAP (EST-coaps). This allows low-resource constrained devices to re-use existing EST functionality. Example low-resource use cases for EST are: secure bootstrapping and certificate enrollment.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 26, 2018.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. EST operational differences	3
1.2. Terminology	4
2. Conformance to RFC7925 profiles	4
3. Protocol Design and Layering	5
3.1. Payload format	6
3.2. Message Bindings	6
3.3. CoAP response codes	6
3.4. Message fragmentation	7
3.5. Deployment limits	8
4. Discovery and URI	8
5. DTLS Transport Protocol	10
6. Proxying	11
7. Parameters	12
8. IANA Considerations	12
8.1. Content-Format registry	12
8.2. Resource Type registry	14
9. Security Considerations	15
9.1. proxy considerations	15
9.2. EST server considerations	15
10. Acknowledgements	16
11. Change Log	16
12. References	17
12.1. Normative References	17
12.2. Informative References	18
Appendix A. EST messages to EST-coaps	20
A.1. cacerts	20
A.2. csrattrs	23
A.3. enroll / reenroll	23
A.4. serverkeygen	25
Appendix B. Encoding for server side key generation	27

Appendix C. EST-coaps Block message examples	27
Authors' Addresses	29

1. Introduction

Enrollment over Secure Transport (EST) [RFC7030] is used for authenticated/authorized endpoint certificate enrollment (and optionally key provisioning) through a Certificate Authority (CA) or Registration Authority (RA). This functionality is also needed for low resource devices.

"Classical" EST uses HTTPS and this specification defines a new transport for EST using CoAP. It also profiles the use of EST to a smaller subset.

IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [RFC4944] on IEEE 802.15.4 [ieee802.15.4] wireless networks are becoming common in many industry application domains such as lighting controls. 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. An example use case is the application of Bootstrapping of Remote Secure Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. The low resource aspects are detailed for 6tisch in [I-D.ietf-6tisch-minimal-security] and [I-D.ietf-6tisch-dtsecurity-secure-join].

Constrained networks use DTLS [RFC6347], CoAP [RFC7252], and UDP instead of TLS [RFC5246], HTTP [RFC7230] and TCP. EST-coaps replaces the invocations of TLS and HTTP by DTLS and CoAP invocations thus enabling EST for CoAP-based low-resource devices.

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.

1.1. 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:

- o Replacement of TLS by DTLS and HTTP by CoAP, resulting in:

- * DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.
- o 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]).

1.2. Terminology

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

2. Conformance to RFC7925 profiles

This section shows how EST-coaps fits into the profiles of low-resource devices as described in [RFC7925].

EST-coaps can transport certificates and private keys. Private keys can be transported as response to a request to a server-side key generation as described in section 4.4 of [RFC7030].

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

- o 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;
- o DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

3. 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 3.4. The Figure 1 below shows the layered EST-coaps architecture.

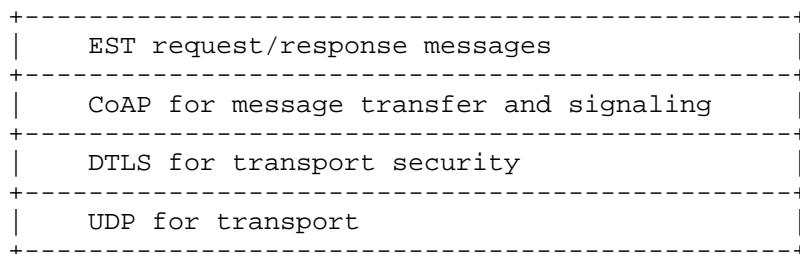


Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design. The parts supported by EST-coaps are identified by their message types:

- o Simple enroll and reenroll, for CA to sign public client-identity key.
- o CA certificate retrieval, needed to receive the complete set of CA certificates.
- o CSR Attributes request messages, informs the client of the fields to include in generated CSR.
- o Server-side key generation messages, to provide a private client-identity key when the client is too restricted or because of lack of an entropy source. [EDNOTE: Encrypting these keys is important. RFC7030 specifies how the private key can be encrypted with CMS using symmetric or asymmetric keys. Mention how symmetric key can be derived for EST server side key generation from the TLS KEM draft.]

3.1. 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 3.3.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple binary 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 binary for all EST-coaps Content-Formats.

3.2. Message Bindings

This section describes the general EST CoAP message characteristics.

It is RECOMMENDED to use CoAP CON messages. This recommendation does not influence the communication efficiency because all EST-coaps messages expect a response.

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 EST REST messages.

EST 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.

3.3. 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 (POST) request, in EST-coaps the equivalent CoAP response code 2.05 (2.01) MUST be used. Response code HTTP 202 in EST is mapped to CoAP ___. In [I-D.hartke-core-pending] it is specified how multiple concurrently

open requests may be handled. 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.

3.4. Message fragmentation

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states that to avoid using IP fragmentation, which involves error-prone datagram reconstitution, invokers of the DTLS record layer SHOULD 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. Section 4.6 of CoAP [RFC7252] 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". Section 4.6 of [RFC7252] also suggests that IPv4 implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes. From [RFC0791] follows that 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. 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 C.

3.5. Deployment limits

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 EST works for networks of 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.

4. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and an additional EST-coaps URI is specified that is shorter than the EST URI.

In the context of CoAP, 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 `"ace.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; throughout this document the example root resource `/est` is used. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=ace.est
```

```
RES: 2.05 Content
</est>; rt="ace.est"
```

The additional EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying a shorter resource path names:

```
coaps://www.example.com/est/short-name
```

The CoAP short URI exists next to the URI defined in [RFC7030].

```
coaps://www.example.com/.well-known/est/est-name
```

OR

```
coaps://www.example.com/.well-known/est/ArbitraryLabel/est-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 URI path to the shorter EST-coaps URI path.

EST	EST-coaps
/cacerts	/crts
/simpleenroll	/sen
/simplereenroll	/sren
/csrattrs	/att
/serverkeygen	/skg

Table 1

When discovering the root path for the EST resources, the server MAY return the full resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. For example, the following more complete response is possible.

```
REQ: GET /.well-known/core?rt=ace.est
```

```
RES: 2.05 Content
</est>; rt="ace.est"
</est/crts>; rt="ace.est";ct=TBD1
</est/sen>; rt="ace.est";ct=TBD1 TBD4
</est/sren>; rt="ace.est";ct=TBD1 TBD4
</est/att>; rt="ace.est";ct=TBD4
</est/skg>; rt="ace.est";ct=TBD1 TBD4 TBD2
```

The return of the content-types allows the client to choose the most appropriate one from multiple content types.

5. DTLS Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the CoAP messages exchanged.

DTLS is one such secure protocol. When "TLS" is referred to in the context of EST, 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].

CoAP and DTLS can provide proof of identity for EST-coaps clients and server with simple PKI messages 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].

Channel-binding information for linking proof-of-identity with connection-based proof-of-possession is optional for EST-coaps. When proof-of-possession is desired, a set of actions are required regarding the use of tls-unique, described in section 3.5 in [RFC7030]. The tls-unique information translates to the contents of the first "Finished" message in the TLS handshake between server and client [RFC5929]. The client is then supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request Info 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 the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message

```
PRF(master_secret, finished_label, Hash(handshake_messages))  
[0..verify_data_length-1];
```

MUST be computed as if each handshake message had been sent as a single fragment [RFC6347].

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.

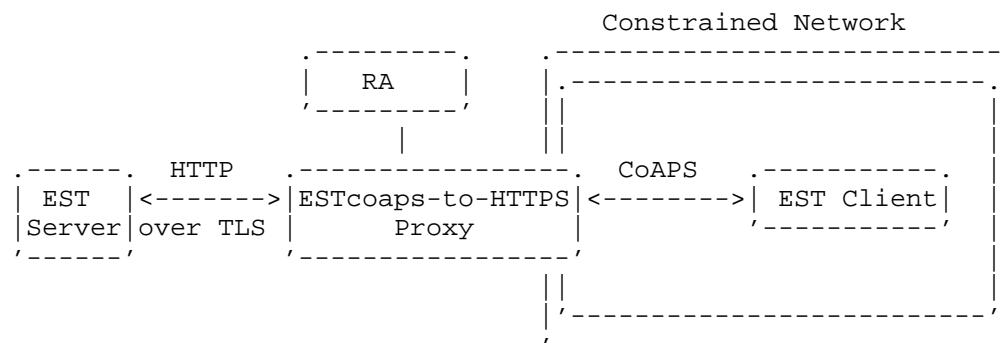
Support for Observe CoAP options [RFC7641] is out-of-scope for this document. Observe options could be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

6. Proxying

In real-world deployments, the EST server will not always reside within the CoAP boundary. The EST-server can exist outside the constrained network in a non-constrained network that supports TLS/HTTP. In such environments EST-coaps is used by the client within the CoAP boundary and TLS is used to transport the EST messages outside the CoAP boundary. A proxy entity at the edge is required to operate between the CoAP environment and the external HTTP network. The ESTcoaps-to-HTTPS proxy SHOULD terminate EST-coaps downstream and initiate EST connections over TLS upstream.

One possible use-case, shown in one figure below, is expected to be deployed in practice:

- o A proxy between any EST-client and EST-server



ESTcoaps-to-HTTPS proxy at the CoAP boundary.

Table 1 contains the URI mapping between the EST-coaps and EST the proxy SHOULD adhere to. Section 7 of [RFC8075] and Section 3.3 define the mapping between EST-coaps and HTTP response codes, that

determines how a proxy translates CoAP response codes from/to HTTP status codes. The mapping from Content-Type to media type is defined in Section 8. The conversion from binary to BSD64 needs to be done in the proxy. Conversion is possible because a TLS link exists between EST-coaps-to-HTTP proxy and EST server and a corresponding DTLS linked exists between EST-coaps-to-HTTP proxy and EST client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP proxy SHOULD reassemble the BLOCKs before translating the binary content to BSD64, and consecutively relay the message upstream into the HTTP environment.

For the discovery of the EST server by the EST client in the coap environment, the EST-coaps-to-HTTP proxy MUST announce itself according to the rules of Section 4. The available functions of the proxies MUST be announced with as many resource paths. The discovery of EST server in the http environment follow the rules specified in [RFC7030].

[EDNOTE: PoP will be addressed here.]

A proxy SHOULD authenticate the client downstream and it should be authenticated by the EST server or CA upstream. The Registration Authority (RA) is necessary to (re-)create the secure connection from DTLS to TLS and vice versa. A trust relationship needs to be pre-established between the proxy and the EST servers to be able to proxy these connections on behalf of various clients.

[EDNOTE: To add more details about trust relations in this section.]

7. Parameters

[EDNOTE: This section to be populated. It will address transmission parameters described in sections 4.7 and 4.8 of the CoAP draft. EST 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 EST. For example, the processing delay of CAs could be less than 2s, but in this case they should send a CoAP ACK every 2s while processing.]

8. IANA Considerations

8.1. Content-Format registry

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
- * ID: TBD1
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5751]
- * Applications that use this media type: EST

2.

- * application/pkcs8
- * Type name: application
- * Subtype name: pkcs8
- * ID: TBD2
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5958]
- * Applications that use this media type: EST

3.

- * application/csrattrs

- * Type name: application
- * Subtype name: csrattrs
- * ID: TBD3
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC7030]
- * Applications that use this media type: EST

4.

- * application/pkcs10
- * Type name: application
- * Subtype name: pkcs10
- * ID: TBD4
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5967]
- * Applications that use this media type: EST

8.2. Resource Type registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

- o rt="ace.est" needs registration with IANA.

9. Security Considerations

9.1. proxy considerations

The proxy proposed in Section 6 must be deployed with great care, and only when the recommended connections are impossible.

[EDNOTE: To add more details about trust relations through proxies in this section.]

9.2. EST server considerations

The security considerations of section 6 of [RFC7030] are only partially valid for the purposes of this document. As HTTP Basic Authentication is not supported, the considerations expressed for using passwords do not apply.

Given that the client has only limited resources and may not be able to generate sufficiently random keys to encrypt its identity, it is possible that the client uses server generated private/public keys to encrypt its certificate. The transport of these keys is inherently risky. A full probability analysis MUST be done to establish whether server side key generation enhances or decreases the probability of identity stealing.

When a client uses the Implicit TA database for certificate validation, the client cannot verify that the implicit data base can act as an RA. It is RECOMMENDED that such clients include "Linking Identity and POP Information" Section 5 in requests (to prevent such requests from being forwarded to a real EST server by a man in the middle). It is RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication be carefully managed to reduce the chance of a third-party CA with poor certification practices from being trusted. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3 of [RFC7030]) limits any vulnerability to the first DTLS exchange.

In accordance with [RFC7030], TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More information about recommendations of TLS and DTLS are included in [RFC7525].

As described in CMC, Section 6.7 of [RFC5272], "For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair". The inclusion of tls-unique in the certification request links the proof-of-possession to

the TLS proof-of-identity. This implies but does not prove that the authenticated client currently has access to the private key.

Regarding the CSR attributes that the CA may list for inclusion in an enrollment request, an adversary could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want. The CA is expected to be able to enforce policies to recover from improper CSR requests.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

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. Constructive comments were received from Eliot Lear, Jim Schaad, Hannes Tschofenig, and Julien Vermillard.

11. Change Log

-03:

removed all motivation to and dependence on BRKI

Supports full EST, except password support

discovery limited to EST functions

/.well-known/est is alternative path to short coap path

proxy discussion is simplified to one case

-02:

binary instead of CBOR binary in mime types.

supported content types are discoverable.

DTLS POP text improved.

First version of Security considerations section written.

First version of Proxying section written.

Various text improvements.

-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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc5751>>.
- [RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <<https://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, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.

- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.

12.2. Informative 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-01 (work in progress), August 2017.
- [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., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-04 (work in progress), October 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-09 (work in progress), October 2017.
- [ieee802.15.4]
Institute of Electrical and Electronics Engineers, "IEEE Standard 802.15.4-2006", 2006.

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [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, <<https://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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<https://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, <<https://www.rfc-editor.org/info/rfc6090>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [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, <<https://www.rfc-editor.org/info/rfc7251>>.

- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://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, <<https://www.rfc-editor.org/info/rfc7925>>.

Appendix A. EST messages to EST-coaps

This section takes all examples from Appendix A of [RFC7030], changes the payload from Base64 to binary and replaces the http headers by their CoAP equivalents.

The corresponding CoAP headers are only shown in Appendix A.1. Creating CoAP headers are assumed to be generally known.

[EDNOTE: The payloads of the examples need to be re-generated with appropriate tools and example certificates.]

A.1. cacerts

In EST-coaps, a coaps cacerts IPv4 message can be:

```
GET coaps://[192.0.2.1:8085]/est/crts
```

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in Appendix C.


```

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 2.05 Content response with a cert in EST-coaps will then be:

```

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 document)

```

```

Payload =
30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a620673410105050030
1b31193017060355040313106573744578616d706c654341204f774f301e170d313
3303530393033353333315a170d3134303530393033353333315a301b3119301706
0355040313106573744578616d706c654341204f774f302062300d06092a6206734
10101050003204f0030204a022041003a923a2968bae4aae136ca4e2512c5200680
358482ac39d6f640e4574e654ea35f48b1e054c5da3372872f7a1e429f4edf39584
32efb2106591d3eb783c1034709f251fc86566bda2d541c792389eac4ec9e181f4b
9f596e5ef2679cc321542b11337f90a44df3c85f1516561fa968a1914f265bc0b82
76ebe3106a790d97d34c8c37c74felc30b396424664ac426284a9f6022e02693843
6880adfc95c98ca1dfc2e6d75319b85d0458de28a9d13fb16d620fff7541f6a25d
7daf004355020301000130b040300f0603551d130101f10530030101fc1d0603551
d0e04160414084d321ca0135e77217a486b686b334b00e0603551d0f0101f104030

```

20106300d06092a62067341010505000320410023703b965746a0c2c978666d787a
94f89b495a11f0d369b28936ec2475c0f0855c8e83f823f2b871a1d92282f323c45
904ba008579216cf5223b8b1bc425a0677262047f7700240631c17f3035d1c3780b
2385241cba1f4a6e98e6be6820306b3a786de5a557795d1893822347b5f825d34a7
ad2876f8feba4d525b31066f6505796f71530003431a3e6bbfe788b4565029a7e20
a51107677552586152d051e8eebf383e92288983421d5c5652a4870c3af74b9bdbe
d6b462e2263d30f6d3020c330206bc20102020101300d06092a6206734101050500
301b31193017060355040313106573744578616d706c654341204f774f301e170d3
133303530393033353333325a170d3134303530393033353333325a301b31193017
060355040313106573744578616d706c654341204e774f302062300d06092a62067
3410101050003204f0030204a02204100ef6b677a3247c1fc03d2b9baf113e5e7e1
1f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a7229283a790
8751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b7bd94338
d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562c4f5abb7
b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c768d03b8
076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c14476c37de0f
55033f192a5ad21f9a2a71c20301000134b050300e0603551d0f0101f104030204c
1d0603551d0e04160414112966e304761732fbfe6a2c823c301f0603551d2304183
0165084d321ca0135e77217a486b686b334b00d06092a6206734101050500032041
00b382ba3355a50e287bae15758b3beff63d34d3e357b90031495d018868e49589b
9faf46a4ad49b1d35b06ef380106677440934663c2cc111c183655f4dc41c0b3401
123d35387389db91f1e1b4131b16c291d35730b3f9b33c7475124851555fe5fc647
e8fd29605367c7e01281bf6617110021b0d10847dce0e9f0ca6c764b6334784055
172c3983d1e3a3a82301a54fcc9b0670c543a1c747164619101ff23b240b2a26394
clf7d38d0e2f4747928ece5c34627a075a8b3122011e9d9158055c28f020c330206
bc20102020102300d06092a6206734101050500301b311930170603550403131065
73744578616d706c654341204e774e301e170d3133303530393033353333325a170
d3134303530393033353333325a301b31193017060355040313106573744578616d
706c654341204f774e302062300d06092a620673410101050003204f0030204a022
041003a923a2968bae4aae136ca4e2512c5200680358482ac39d6f640e4574e654e
a35f48b1e054c5da3372872f7a1e429f4edf3958432efb2106591d3eb783c103470
9f251fc86566bda2d541c792389eac4ec9e181f4b9f596e5ef2679cc321542b1133
7f90a44df3c85f1516561fa968a1914f265bc0b8276ebe3106a790d97d34c8c37c7
4felc30b396424664ac426284a9f6022e026938436880adfc95c98ca1dfc2e6d75
319b85d0458de28a9d13fb16d620ffff7541f6a25d7daf004355020301000134b050
300e0603551d0f0101f104030204c1d0603551d0e04160414084d321ca0135e7721
7a486b686b334b01f0603551d230418301653112966e304761732fbfe6a2c823c30
0d06092a6206734101050500032041002e106933a443070acf5594a3a584d08af7e
06c295059370a06639eff9bd418d13bc25a298223164a6cf1856b11a81617282e4a
410d82ef086839c6e235690322763065455351e4c596acc7c016b225dec094706c2
a10608f403b10821984c7c152343b18a768c2ad30238dc45dd653ee6092b0d5cd4c
2f7d236043269357f76d13f95fb5f00d0e19263c6833948e1ba612ce8197af650e2
5d882c12f4b6b9b67252c608ef064aca3f9bc867d71172349d510bb7651cd438837
73d927deb41c4673020bb302063c201020209009b9dda3324700d06092a62067341
01050500301b31193017060355040313106573744578616d706c654341204e774e3
01e170d3133303530393033353333325a170d3134303530393033353333325a301b
31193017060355040313106573744578616d706c654341204e774e302062300d060
92a620673410101050003204f0030204a02204100ef6b677a3247c1fc03d2b9baf1

```
13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
551d0f0101f10403020106300d06092a620673410105050003204100423f06d4b76
0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
34119794b90e320bc730d6c1bee496e7ac125ce9aleca595a3a4c54a865e6b623c9
247bfd0a7c19b56077392555c955e233642bec643ae37c166c5e221d797aea3748f
0391c8d692a5cf9bb71f6d0e37984d6fa673a30d0c006343116f58403100
```

A.2. csrattrs

In the following valid /csrattrs exchange, the EST-coaps client authenticates itself with a certificate issued by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The IPv6 CoAP GET request looks like:

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att
```

A 2.05 Content response contains attributes which are relevant for the authenticated client. In this example, the EST-coaps server two attributes that the client can ignore when they are unknown to him.:

A.3. enroll / reenroll

[EDNOTE: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]

During the Enroll/Reenroll exchange, the EST-coaps client uses a CSR (PKCS#10) request in the POST request payload.

After verification of the certificate by the server, a 2.05 Content response with the issued certificate will be returned.

POST [2001:db8::2:1]:61616/est/sen
(Content-Format: application/pkcs10)
30208530206d020100301f311d301b0603550403131464656d6f7374657034203
1333638313431333532302062300d06092a620673410101050003204f0030204a
022041005d9f4dffd3c5949f646a9584367778560950b355c35b8e34726dd3764
54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
b871691e0f02030100018701f06092a620673410907311213102b72724369722f
372b45597535305434300d06092a620673410105050003204100441b40177a3a6
5501487735a8ad5d3827a4eaa867013920e2afcd87aa81733c7c0353be47e1bf
a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
17071103407553155cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40

RET:

2.05 Content (Content-Format: application/pkcs7-mime)
3020f806092a62067341070283293020e50201013100300b06092a62067341070
1830b3020c730206fc20102020115300d06092a6206734101050500301b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
0393233313535335a170d3134303530393233313535335a301f311d301b060355
0403131464656d6f73746570342031333638313431333532302062300d06092a6
20673410101050003204f0030204a022041005d9f4dffd3c5949f646a95843677
78560950b355c35b8e34726dd376454231734795b4c09b9c6d75d408311307a81
f7adef7f5d241f7d5be85620c5d4438bbb4242cf215c167f2ccf36c364ea2618a
62f0536576369d6304e6a968772247d86824f079faac7a6f694cfda5b84c42087
dc062d462190c525813f210a036a737b4f30d8891f4b75559fb72752453146332
d51c937557716ccec624f5125c3a4447ad3115020048113fef54ad554ee88af09
a2583aac9024075113db4990b1786b871691e0f020301000134b050300e060355
1d0f0101f104030204c1d0603551d0e04160414e81d0788aa2710304c5ecd4d1e
065701f0603551d230418301653112966e304761732fbfe6a2c823c300d06092a
6206734101050500032041002910d86f2ffeeb914c046816871de601567d291b4
3fabee0f0e8ff81cea27302a7133e20e9d04029866a8963c7d14e26fbe8a0ab1b
77fbb1214bbcdc906fbc381137ec1de685f79406c3e416b8d82f97174bc691637
5a4e1c4bf744c7572b4b2c6bade9fb35da786392ee0d95e3970542565f3886ad6
7746d1b12484bb02616e63302dc371dc6006e431fb7c457598dd204b367b0b3d3
258760a303f1102db26327f929b7c5a60173e1799491b69150248756026b80553
171e4733ad3d13c0103100

[EDNOTE: If POP is used, make sure tls-unique in the CSR is a valid
HMAC output.]

A.4. serverkeygen

During this valid /serverkeygen exchange, the EST-coaps client authenticates itself using the certificate provided by the connected CA.

[EDNOTE: the client includes a CSR with a public key that the server should ignore, so we need a content-format here.]

[EDNote: If POP is used, make sure tls-unique in the CSR is a valid HMAC output.]

The initial DTLS handshake is identical to the enrollment example. The CoAP GET request looks like:

```
POST coaps://[192.0.2.1:8085]/est/skg
302081302069020100305b313e303c060355040313357365727665724b6579476
56e2072657120627920636c69656e7420696e2064656d6f207374657020313220
3133363831343139353531193017060355040513105049443a576964676574205
34e3a3130302062300d06092a620673410101050003204f0030204a02204100f4
dfa6c03f7f2766b23776c333d2c0f9d1a7a6ee36d01499bbe6f075d1e38a57e98
ecc197f51b75228454b7f19652332de5e52e4a974c6ae34e1df80b33f15f47d3b
cbf76116bb0e4d3e04a9651218a476a13fc186c2a255e4065ff7c271cff104e47
31fad53c22b21a1e5138bf9ad0187314ac39445949a48805392390e78c7659621
6d3e61327a534f5ea7721d2b1343c7362b37da502717cfc2475653c7a3860c5f4
0612a5db6d33794d755264b6327e3a3263b149628585b85e57e42f6b3277591b0
2030100018701f06092a6206734109073112131064467341586d4a6e6a6f6b427
4447672300d06092a620673410105050003204100472d11007e5a2b2c2023d47a
6d71d046c307701d8ebc9e47272713378390b4ee321462a3dbe54579f5a514f6f
4050af497f428189b63655d03a194ef729f101743e5d03fbc6ae1e84486d1300a
f9288724381909188c851fa9a5059802eb64449f2a3c9e441353d136768da27ff
4f277651d676a6a7e51931b08f56135a2230891fd184960e1313e7a1a9139ed19
28196867079a456cd2266cb754a45151b7b1b939e381be333fea61580fe5d25bf
4823dbd2d6a98445b46305c10637e202856611
```

RET:

```
2.05 Content (Content-Format: application/pkcs8)
30213e020100300d06092a6206734101010500042128302124020100022041003
c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274
dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1
1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c
0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7
```

```
f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef
7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
241366b7907187583b511102103369047e5cce0b65012200df5ec697b5827575c
db6821fff299d6a69574b31ddf0f9e245ea2f74396c24b3a7565067e41366423b
5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
173de2c3aaf200af535411c7090210401421e2ea217e37312dcc606f453a6634
f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
f917438f607f18181684376e13a39e07
--estServerExampleBoundary
3020c506092a62067341070283363020f20201013100300b06092a62067341070
183183020d430207cc20102020116300d06092a6206734101050500301b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
039323323535365a170d3134303530393233323535365a302c312a3028060355
0403132173657276657273696465206b65792067656e657261746564207265737
06f6e7365302062300d06092a620673410101050003204f0030204a022041003c
0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274d
d01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a11
bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c0
c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d54
5e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704e
fec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e310830100
0134b050300e0603551d0f0101f104030204c1d0603551d0e04160414764b1bd5
e69935626e476b195a1a8c1f0603551d230418301653112966e304761732fbfe6
a2c823c300d06092a620673410105050003204100474e5100a9cdaaa813b30f48
40340fb17e7d6d6063064a5a7f2162301c464b5a8176623dfb1a4a484e618delc
3c3c5927cf590f4541233ff3c251e772a9a3f2c5fc6e5ef2fe155e5e385deb846
b36eb4c3c7ef713f2d137ae8be4c022715fd033a818d55250f4e6077718180755
a4fa677130da60818175ca4ab2af1d15563624c51e13dfdcf381881b72327e2f4
9b7467e631a27b5b5c7d542bd2edaf78c0ac294f3972278996bdf673a334ff74c
84aa7d65726310252f6a4f41281ec10ca2243864e3c5743103100
```

Without the DecryptKeyIdentifier attribute, the response has no additional encryption beyond DTLS. [EDNOTE: Add comment about deriving symmetric keys by using the TLS KEM draft.]

The response contains first a preamble that can be ignored. The EST-coaps server can use the preamble to include additional explanations, like ownership or support information

Appendix B. Encoding for server side key generation

Sever side key generation for CoAP can be implemented efficiently using multipart encoding

[EDNOTE: text to be written.]

Appendix C. 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 2509 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 39 packets with a payload of 64 bytes each, followed by a packet of 13 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 40 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 -->
    <-- (2:0/1/39) 2.05 Content
GET URI (2:1/1/39) -->
    <-- (2:1/1/39) 2.05 Content
    |
    |
    |
GET URI (2:65/1/39) -->
    <-- (2:65/0/39) 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:

```
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 =
30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a6206734101
```

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 = 0x1A (block number = 1, M=1, SZX=2)
Payload =
05050030
1b31193017060355040313106573744578616d706c654341204f774f301e170d313
3303530393033353333315a170d3134303530393033353333315a
```

The 40th 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 = 0x272 (block number = 39, M=0, SZX=2)
Payload = 73a30d0c006343116f58403100
```

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: <http://www.sandelman.ca/>

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se