ACE Working Group S. Aragon
Internet-Draft M. Tiloca
Intended status: Standards Track S. Raza
Expires: May 2, 2018 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

Copyright Notice

Table of Contents

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.

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

- 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.

- A Security Parameter Index (SPI);

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

- The Security Parameters MUST always include the set of parameters sec_A shown in Figure 2.
- 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 en 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.

```
             C                                  RS                    AS
             |                                   |                     |
             | [------ Resource Request ------>] |                     |
     (1)     |                                   |                     |
             | [<------ AS Information --------] |                     |
             |                                   |                     |
     ---     |                                   |                     |
             |                                   |                     |
             | --------- Token Request ------------------------------> |
     (2)     |                                   |                     |
             |                                                         |
             |              Access Token + RS Information              |
             | <------------------------------------------------------ |
             |     Including information for IPsec SA establishment    |
             |                                                         |
     ---     |                                   |                     |
             |                                   |                     |
             | --------- Access Token ---------> |                     |
             |                                   |                     |
             | [<=== IPsec SA establishment ==>] |                     |
     (3)     |                                   |                     |
             | ======== Resource Request ======> |                     |
             |                                   |                     |
             | <======= Resource Response ====== |                     |
             |                                   |                     |
```

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&=1GFfg ...
    (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'+a+Dg2jjU+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:

- a symmetric key to be used as proof-of-possession key;

- a key identifier associated to the symmetric key;

- 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&=1GFfg ...
    (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:

- In case of Direct Provisioning, the "ipsec" structure is present, while the "COSE_Key" object is not present.

- 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".

- 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.

```
                 Resource
      Client     Server
      |          |
      |          |
      +--------->| Header: POST (Code=0.02)
      | POST     | Uri-Path:"authz-info"
      |          | Content-Type: application/cbor
      |          | Payload: Access Token
      |          |
      |<---------+ IKE_SA_INIT
      |          |
           ...
```

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

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

5.2. Confirmation Methods registration

5.2.1. IPsec field

- Confirmation Method Name: "ipsec"

- Confirmation Method Value: TBD

- Confirmation Method Description: A structure containing the corresponding information of an IPsec Security Association Pair.

- Change Controller: IESG

- Specification Document: This document

5.2.2. Key Management Protocol field

- Confirmation Method Name: "kmp"

- Confirmation Method Value: TBD

- Confirmation Method Description: Key management protocol.

- Change Controller: IESG

- 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

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

```
                               OSCORE context &
                               SA AH-transport
       . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
       .                                                           .
       .         SA ESP-tunnel                                     .
       .  . . . . .  . . . . . . . . .                             .
       .  .                          .                             .
       v  v                          v                             v
   +--------+                    +--------+                    +------+
   | Client |  <==== (1) ====>   | Proxy  |   <==== (2) ====>  |  RS  |
   +--------+                    +--------+                    +------+


 (1): |IP:P|ESP|IP:RS|AH|UDP|OSCORE|ESP_T|ESP_Auth|
 (2): |IP:RS|AH|UDP|OSCORE|
```

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.

```
                   Resource
         Client    Server
         |         |
         |         |
         +-------->| Header: POST (Code=0.02)
         |  POST   | Uri-Path:"authz-info"
         |         | Content-Type: application/cbor
         |         | Payload: EDHOC message_1 + Access Token
         |         |
             ...
```

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 L. Seitz
Internet-Draft RISE SICS
Intended status: Informational S. Erdtman
Expires: May 3, 2018 Spotify AB
October 30, 2017

Raw-Public-Key and Pre-Shared-Key as OAuth client credentials
draft-erdtman-ace-rpcc-02

Abstract

This document describes Transport Layer Security (TLS) authentication using Raw-Public-Key and Pre-Shared-Key as new mechanisms for OAuth client authentication. Although defined for TLS the mechanisms are equally applicable for DTLS.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes Transport Layer Security (TLS) authentication using Raw-Public-Key and Pre-Shared-Key as the mechanism for OAuth client authentication. Examples of endpoint requiring client authentication are token and introspection.

The OAuth 2.0 Authorization Framework [RFC6749] defines a shared secret method of client authentication but also allows for the definition and use of additional client authentication mechanisms when interacting with the authorization server’s token endpoint. This document describes two additional mechanisms of client authentication utilizing Raw-Public-Key [RFC7250] and Pre-Shared-Key TLS [RFC4279], which provide better security characteristics than shared secrets.

To get most bennefits and improved security with these new client credential types it is recomended to use the ’one credential per Client Software Instance’ paradigm. This can be achived by letting the client dynamicly register as described in [RFC7591].

1.1. Requirements Language

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

2. Pre-Shared-Key for Client Authentication

The following section defines, as an extension of OAuth 2.0, Section 2.3 [RFC6749], using Pre-Shared-Key with TLS [RFC4279] to authenticate the client. This method is registered as 'tls_client_psk' in "OAuth Token Endpoint Authentication Methods" registry. If this method is to be used, the client and the Authorization Server MUST share a secret key, and they MUST agree on an identifier for this key.

The (D)TLS handshake MUST be done according to [RFC4279], with the client indicating support for one or more Pre-Shared-Key cipher suites and authorization server selecting a Pre-Shared-Key cipher suite. In order to enable the authorization server to select the correct pre-shared-key the client MUST send the key identifier in the psk-identity field of the ClientKeyExchange message. How the authorization server maps the identifier to a pre-shared-key, and to a specific client is out of scope for this specification.

Note that the key identifier MUST be 2^16 bytes or shorter, in order to fit into the psk-identity field.

3. Raw-Public-Key for Client Authentication

The following section defines, as an extension of OAuth 2.0, Section 2.3 [RFC6749], the use of Raw-Public-Key with (D)TLS [RFC7250] to authenticate the client. This method is registered as 'tls_client_rpk' in "OAuth Token Endpoint Authentication Methods" registry.

The (D)TLS handshake MUST be done according to [RFC7250], with the client indicating support for Raw-Public-Key certificates and the authorization server asking client send its Raw Public Key certificate. Since the client cannot send an explicit client or key identifier in the handshake, the authorization server MUST derive a client identifier from RPK that the client uses.

Note to implementers: Authorization servers can use the following method to map a Raw Public Key to a client identifier: The client identifier is generated from the Raw Public Key using the procedure specified in section 3 of [RFC6920]. The digest is calculated on the Raw Public Key only (not on the SubjectPublicKeyInfo used in the handshake). An example is shown in Figure 1.

```
            Raw Public Key (Base64 encoded):
            MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEEtboxNKPgxEKV9JTNzy
            tUvAbxEfkCTVB9kOzheF5wRAoOz2NKP+ln+XLVAQSp1D6jfo09tppvN
            poQA1nnBNH6A==";

            Encoding:
            ni:///sha-256;xzLa24yOBeCkos3VFzD2gd83Urohr9TsXqY9nhdDN0
```

Figure 1: Example encoding of a raw public key in the Named Information URI Format

4. Dynamic Registration

For dynamic registration of a RPK this specification registers the new parameter ’rpk’ to the Client Registration Metadata Registry. When used this parameter MUST contain a JSON Web Key representing the public key of the client. When ’rpk’ is present in the registration request ’token_endpoint_auth_method’ MUST include ’tls_client_rpk’.

For dynamic registration of a PSK this specification registers the new parameter ’psk’ to the Client Registration Metadata Registry. When used this parameter MUST contain a JSON Web Key representing the key of the client. When registering the client can include the key in the registrations request or the authorisation can generate the key and return it. If the ’psk’ attribute is present in a request ’token_endpoint_auth_method’ MUST include ’tls_client_psk’. To request the authorisation server to generate the key the client includes ’tls_client_psk’ in ’token_endpoint_auth_method’ but does not send ’psk’ attribute.

The ’jwks’ and ’jwks_uri’ is not used to avoid conflict and confusion with application layer keys.

5. Acknowledgements

This document is highly inspired by [I-D.ietf-oauth-mtls] written by B. Campbell, J. Bradley, N. Sakimura and T. Lodderstedt.

6. IANA Considerations

6.1. OAuth Dynamic Client Registration Metadata Registration

This specification requests registration of the following value in the IANA "OAuth Dynamic Client Registration Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591].

6.1.1. Registry Contents

- Client Metadata Name: "rpk"
- Client Metadata Description: JWK for client Raw-Public-Key, can be included in request.
- Change Controller: IESG
- Specification Document(s): [[this specification]]

- Client Metadata Name: "psk"
- Client Metadata Description: JWK for client Pre-Shared-Key, can be included both in request and response.
- Change Controller: IESG
- Specification Document(s): [[this specification]]

6.2. Token Endpoint Authentication Method Registration

This specification requests registration of the following value in the IANA "OAuth Token Endpoint Authentication Methods" registry [IANA.OAuth.Parameters] established by [RFC7591].

6.2.1. Registry Contents

- Token Endpoint Authentication Method Name: "tls_client_rpk"
- Change Controller: IESG
- Specification Document(s): [[this specification]]

- Token Endpoint Authentication Method Name: "tls_client_psk"
- Change Controller: IESG
- Specification Document(s): [[this specification]]

7. Security Considerations

TBD

8. References

8.1. 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>.

[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>.

[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>.

[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>.

[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>.

[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>.

8.2. Informative References

[I-D.ietf-oauth-mtls]
Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "Mutual TLS Profile for OAuth 2.0", draft-ietf-oauth-mtls-04 (work in progress), October 2017.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

Authors' Addresses

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig.seitz@ri.se

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

Email: erdtman@spotify.com

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 will expire on September 20, 2018.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

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

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

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

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

- Claim Key: 6
- Claim Value Type(s): integer or floating-point number
- Change Controller: IESG
- Specification Document(s): Section 3.1.6 of [[this specification]]

- Claim Name: "cti"
- Claim Description: CWT ID
- JWT Claim Name: "jti"
- Claim Key: 7
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): Section 3.1.7 of [[this specification]]

9.2. Media Type Registration

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

9.2.1. Registry Contents

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

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

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

9.3. CoAP Content-Formats Registration

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

9.3.1. Registry Contents

- Media Type: application/cwt
- Encoding: -
- Id: TBD (maybe 61)
- Reference: [[this specification]]

9.4. CBOR Tag registration

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

9.4.1. Registry Contents

- CBOR Tag: TBD (maybe 61 to use the same value as the Content-Format)
- Data Item: CBOR Web Token (CWT)
- Semantics: CBOR Web Token (CWT), as defined in [[this specification]]
- Description of Semantics: [[this specification]]
- 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.MediaTypes]
IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <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
ca69ed8919a394d42f0f2001010202524173796d6d657472696345434453413132
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'4173796d6d65747269634543445341
              3235362' / '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
45f7e6afcda1aae5f6495830c58627087fc5b4974f319a8707a635dd643b
```

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
                     a80f27562f400ee1d0d6c0fdb559c02421fd384fc2e
                     be22d7071378b0ea7428fff157444d45f7e6afcda1a
                     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
610a7e0f677b7e9b0bc73adefdcee16d9d5d284c616abeab5d8c291ce0
```

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
                     f142631610a7e0f677b7e9b0bc73adefdcee16d9d5d
                     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.

```
d18443a10104a1044c53796d6d65747269633235364ba106fb41d584367c2000
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.

- Clarified the relationships between the JWT and CWT "NumericDate" and "StringOrURI" terms, as suggested by Adam Roach.

- Eliminated unnecessary uses of the word "type", as suggested by Adam Roach.

- 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.

- Removed a superfluous comma, as suggested by Warren Kumari.

- Acknowledged additional reviewers.

-13

- Clarified the registration criteria applied to different ranges of Claim Key values, as suggested by Kathleen Moriarty and Dan Romascanu.

- No longer describe the syntax of CWT claims as being the same as that of the corresponding JWT claims, as suggested by Kyle Rose.

- Added guidance about the selection of the Designated Experts, as suggested by Benjamin Kaduk.

- Acknowledged additional reviewers.

-12

- Updated the RFC 5226 reference to RFC 8126.

- Made the IANA registration criteria consistent across sections.

- 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.

- Changed the "Reference" field name to "Description of Semantics" in the CBOR Tag registration request.

- 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

- Corrected the "iv" value in the signed and encrypted CWT example.

- Mention CoAP in the "application/cwt" media type registration.

- 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.

- Updated Acknowledgements.

-10

- 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.

- Clarified the nested CWT description.

- Changed uses of "binary string" to "byte string".

-09

- Added key ID values to the examples.

- Key values for the examples are now represented in COSE_Key format using CBOR diagnostic notation.

-08

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

-07

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

-06

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

-05

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

- Say that CWT is derived from JWT, rather than CWT is a profile of JWT.

- Used CBOR type names in descriptions, rather than major/minor type numbers.

- Clarified the NumericDate and StringOrURI descriptions.

- Changed to allow CWT claim names to use values of any legal CBOR map key type.

- Changed to use the CWT tag to identify nested CWTs instead of the CWT content type.

- Added an example using a floating-point date value.

- Acknowledged reviewers.

-04

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

-03

- Reworked the examples to include signed, MACed, encrypted, and nested CWTs.

- Defined the CWT CBOR tag and explained its usage.

-02

- Added IANA registration for the application/cwt media type.

- Clarified the nested CWT language.

- Corrected nits identified by Ludwig Seitz.

-01

- Added IANA registration for CWT Claims.

- Added IANA registration for the application/cwt CoAP content-format type.

- Added Samuel Erdtman as an editor.

- Changed Erik’s e-mail address.

-00

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

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

March 05, 2018

Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)
draft-ietf-ace-dtls-authorize-03

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

The DTLS RPK handshake [RFC7250] requires client authentication to provide proof-of-possession for the key tied to the access token. Here the access token needs to be transferred to the resource server before the handshake is initiated, as described in section 5.8.1 of draft-ietf-ace-oauth-authz [1].

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

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

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

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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].

2. Protocol Overview

The CoAP-DTLS profile for ACE specifies the transfer of authentication 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 a 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 RS.

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

```
      C                             RS                   AS
      | [-- Resource Request --->] |                     |
      |                            |                     |
      | [<----- AS Information --] |                     |
      |                            |                     |
      | --- Token Request  ----------------------------> |
      |                            |                     |
      | <---------------------------- Access Token ----- |
      |                               + RS Information   |
```

Figure 1: Retrieving an Access Token

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

Once the client C knows the authorization server's address, it can send an Access Token request to the token endpoint at the AS as specified in [I-D.ietf-ace-oauth-authz]. 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. How the mutual authentication between the client and the authorization server is achieved is out of scope for this document; the client may have been configured with a public key of the authorization server and have been registered at the AS via the OAuth client registration mechanism as outlined in section 5.3 of draft-ietf-ace-oauth-authz [2].

If C wants to use the CoAP RawPublicKey mode as described in Section 9 of RFC 7252 [3] it MUST provide a key or key identifier within a "cnf" object in the token request. If the authorization server AS decides that the request is to be authorized it generates an access token response for the client C containing a "profile" parameter with the value "coap_dtls" to indicate that this profile MUST be used for communication between the client C and the resource server. Is also adds a "cnf" parameter with additional data for the establishment of a secure DTLS channel between the client and the resource server. The semantics of the 'cnf' parameter depend on the type of key used between the client and the resource server and control whether the client must use RPK mode or PSK mode to establish a DTLS session with the resource server, see Section 3 and Section 4.

The Access Token returned by the authorization server then can be used by the client to establish a new DTLS session with the resource server. When the client intends to use asymmetric cryptography in the DTLS handshake with the resource server, the client MUST upload the Access Token to the authz-info resource on the resource server before starting the DTLS handshake, as described in section 5.8.1 of draft-ietf-ace-oauth-authz [4]. If only symmetric cryptography is used between the client and the resource server, the Access Token MAY instead be transferred in the DTLS ClientKeyExchange message (see Section 4.1).

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

```
   C                            RS                   AS
   | [--- Access Token ------>] |                     |
   |                            |                     |
   | <== DTLS channel setup ==> |                     |
   |                            |                     |
   | == Authorized Request ===> |                     |
   |                            |                     |
   | <=== Protected Resource == |                     |
```

Figure 2: Protocol overview

The following sections specify how CoAP is used to interchange access-related data between the resource server 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.

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

2.1. Resource Access

Once a DTLS channel has been established as described in Section 3 and Section 4, 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.

On the resource server side, successful establishment of the DTLS channel binds the client to the access token, functioning as a proof-of-possession associated key. Any request that the resource server receives on this channel MUST be checked against these authorization rules that are associated with the identity of the client. 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 as described in Section 5.1.1 of draft-ietf-ace-oauth-authz [5].

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

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

The resource server SHOULD treat an incoming CoAP request as authorized if the following holds:

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

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

3. The request is destined for 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 MUST be rejected according to [Section 5.1.1 of draft-ietf-ace-oauth-authz](https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.1

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 cannot always know a priori if an Authorized Resource Request will succeed. If the client repeatedly gets error responses containing AS Information (cf. Section 5.1.1 of draft-ietf-ace-oauth-authz [6] 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.

2.2. Dynamic Update of Authorization Information

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

Figure 3 depicts the message flow where the client C requests a new Access Token after a security association between the client and the resource server RS has been established using this protocol. The token request MUST specify the key identifier of 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 ticket that has previously been issued to the requesting client. Otherwise, the Client-to-AS request MUST be declined with a the error code "unsupported_pop_key" as defined in Section 5.6.3 of draft-ietf-ace-oauth-authz [7].

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 associate the updated authorization information with any existing DTLS session that is identified by this key identifier.

Note: 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.

```
      C                            RS                   AS
      | <===== DTLS channel =====> |                     |
      |        + Access Token      |                     |
      |                            |                     |
      | --- Token Request  ----------------------------> |
      |                            |                     |
      | <---------------------------- New Access Token - |
      |                              + RS Information    |
      |                            |                     |
      | --- Update /authz-info --> |                     |
      |     New Access Token       |                     |
      |                            |                     |
      | == Authorized Request ===> |                     |
      |                            |                     |
      | <=== Protected Resource == |                     |
```

Figure 3: Overview of Dynamic Update Operation

2.3. Token Expiration

DTLS sessions that have been established in accordance with this profile are always tied to a specific set of access tokens. As these tokens may become invalid at any time (either because the token has expired or the responsible authorization server has revoked the token), the session may become useless at some point. A resource server therefore may decide to terminate existing DTLS sessions after the last valid access token for this session has been deleted.

As specified in section 5.8.2 of draft-ietf-ace-oauth-authz [8], the resource server MUST notify the client with an error response with code 4.01 (Unauthorized) for any long running request before terminating the session.

The resource server MAY also keep the session alive for some time and respond to incoming requests with a 4.01 (Unauthorized) error message including AS Information to signal that the client needs to upload a new access token before it can continue using this DTLS session. The AS Information is created as specified in section 5.1.2 of draft-ietf-ace-oauth-authz [9]. The resource server SHOULD add a "kid" parameter to the AS Information denoting the identifier of the key that it uses internally for this DTLS session. The client then includes this "kid" parameter in a Client-to-AS request used to retrieve a new access token to be used with this DTLS session. In case the key identifier is already known by the client (e.g. because it was included in the RS Information in an AS-to-Client response), the "kid" parameter MAY be elided from the AS Information.

Table 1 updates Figure 2 in section 5.1.2 of draft-ietf-ace-oauth-authz [10] with the new "kid" parameter in accordance with [RFC8152].

Parameter name	CBOR Key	Major Type
kid	4	2 (byte string)

Table 1: Updated AS Information parameters

3. RawPublicKey Mode

To retrieve an access token for the resource that the client wants to access, the client requests an Access Token from the authorization server. The client MUST add a "cnf" object carrying either its raw public key or a unique identifier for a public key that it has previously made known to the authorization server. To prove that the client is in possession of this key, it MUST use the same public key as in certificate message that is used to establish the DTLS session with the authorization server.

An example Access Token request from the client to the resource server is depicted in Figure 4.

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

Figure 4: Access Token Request Example for RPK Mode

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

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

An implementation that supports the RPK mode of this profile MUST at least support the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251] with the ed25519 curve (cf. [RFC8032], [I-D.ietf-tls-rfc4492bis]).

Note: According to [RFC7252], CoAP implementations MUST support the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251] and the NIST P-256 curve. As discussed in [RFC7748], new ECC curves have been defined recently that are considered superior to the so-called NIST curves. The curve that is mandatory to implement in this specification is said to be efficient and less dangerous regarding implementation errors than the secp256r1 curve mandated in [RFC7252].

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. If CBOR web tokens [I-D.ietf-ace-cbor-web-token] are used as recommended in [I-D.ietf-ace-oauth-authz], the authorization server MUST include a "COSE_Key" object in the "cnf" claim of the Access Token. This "COSE_Key" object MAY contain a reference to a key for the client that is already known by the resource server (e.g., from previous communication). 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.

4. PreSharedKey Mode

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 session key to construct the proof-of-possession token and therefore MUST specify a symmetric key that was previously generated by the authorization server as a session key for the communication between the client and the resource server.

Depending on the requested token type and algorithm in the Access Token request, the authorization server adds RS Information to the response that provides the client with sufficient information to setup a DTLS channel with the resource server. For symmetric proof-of-possession keys (c.f. [I-D.ietf-ace-oauth-authz]), the client must ensure that the Access Token request is sent over a secure channel that guarantees authentication, message integrity and confidentiality.

When the authorization server authorizes the client it returns an AS-to-Client response with the profile parameter set to "coap_dtls" and a "cnf" parameter carrying a "COSE_Key" object that contains the symmetric session key to be used between the client and the resource server as illustrated in Figure 5.

```
2.01 Created
Content-Format: application/cbor
Location-Path: /token/asdjbaskd
Max-Age: 86400
{
   access_token: h’d08343a10...
   (remainder of CWT omitted for brevity)
   token_type:   pop,
   alg:          HS256,
   expires_in:   86400,
   profile:      coap_dtls,
   cnf: {
     COSE_Key: {
       kty: symmetric,
       k: h’73657373696f6e6b6579’
     }
   }
}
```

Figure 5: Example Access Token response

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

A response that declines any operation on the requested resource is constructed according to Section 5.2 of RFC 6749 [11], (cf. Section 5.7.3 of [I-D.ietf-ace-oauth-authz]).

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

Figure 6: Example Access Token response with reject

4.1. DTLS Channel Setup Between C and RS

When a client receives an Access Token from an authorization server, it checks if the payload contains an "access_token" parameter and a "cnf" parameter. With this information the client can initiate 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.

In PreSharedKey mode, the knowledge of the session key 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 session key 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. Alternatively, the client MAY 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 and not perform any re-coding.

Note: As stated in section 4.2 of [RFC7925], the PSK identity should be treated as binary data in the Internet of Things space and not assumed to have a human-readable form of any sort.

If a resource server receives a ClientKeyExchange message that contains a "psk_identity" with a length greater zero, it uses the contents as index for its key store (i.e., treat the contents as key identifier). The resource server MUST check if it has one or more Access Tokens that are associated with the specified key. If no valid Access Token is available for this key, the DTLS session setup is terminated with an "illegal_parameter" DTLS alert message.

If no key with a matching identifier is found the resource server the resource server MAY process the decoded contents of the "psk_identity" field as access token that is stored with the authorization information endpoint before continuing the DTLS handshake. If the decoded contents of the "psk_identity" do not yield a valid access token for the requesting client, the DTLS session setup is terminated with an "illegal_parameter" DTLS alert message.

Note1: As a resource server cannot provide a client with a meaningful PSK identity hint in
response to the client’s ClientHello message, the resource server SHOULD NOT send a ServerKeyExchange message.

Note2: According to [RFC7252], CoAP implementations MUST support the ciphersuite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]. A client is therefore expected to offer at least this ciphersuite to the resource server.

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

While the client can retrieve the session key 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 session key when no explicit "kid" was provided in the "psk_identity" field. Usually, this is done by including a "COSE_Key" object carrying either a key that has been encrypted with a shared secret between the authorization server and the resource server, or a key identifier that can be used by the resource server to lookup the session key.

Instead of the "COSE_Key" object, the authorization server MAY include a "COSE_Encrypt" structure to enable the resource server to calculate the session key from the Access Token. The "COSE_Encrypt" structure MUST use the _Direct Key with KDF_ method as described in Section 12.1.2 of RFC 8152 [12]. The authorization server MUST include a Context information structure carrying a PartyU "nonce" parameter carrying the nonce that has been used by the authorization server to construct the session key.

This specification mandates that at least the key derivation algorithm "HKDF SHA-256" as defined in [RFC8152] MUST be supported. This key derivation function is the default when no "alg" field is included in the "COSE_Encrypt" structure for the resource server.

4.2. Updating Authorization Information

Usually, the authorization information that the resource server keeps for a client is updated by uploading a new Access Token as described in Section 2.2.

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

After successful completion of the DTLS handshake the resource server updates the existing authorization information for the client according to the new Access Token.

5. 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 and privacy considerations from section 6 and section 7 also apply to this profile.

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.

[I-D.tiloca-tls-dos-handshake] specifies a TLS extension to prevent this type of attack which is applicable especially for constrained environments where the authorization server can act as trust anchor.

6. Privacy Considerations

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 the 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.

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

7. 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: 1

Change Controller: IESG

Reference: [RFC-XXXX]

8. References

8.1. 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-10 (work in progress), February 2018.

[I-D.tiloca-tls-dos-handshake]
Tiloca, M., Seitz, L., Hoeve, M., and O. Bergmann, "Extension for protecting (D)TLS handshakes against Denial of Service", draft-tiloca-tls-dos-handshake-01 (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>.

[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>.

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <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>.

8.2. Informative References

[I-D.ietf-ace-cbor-web-token]
Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-12 (work in progress), February 2018.

[I-D.ietf-tls-rfc4492bis]
Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", draft-ietf-tls-rfc4492bis-17 (work in progress), May 2017.

[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>.

[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>.

[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>.

8.3. URIs

[1] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.8.1

[2] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.3

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

[4] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.8.1

[5] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.1

[6] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.1

[7] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.6.3

[8] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.8.2

[9] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.2

[10] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.2

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

[12] https://tools.ietf.org/html/rfc8152#section-12.1.2

Authors’ Addresses

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

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

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

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

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

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

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

ACE Working Group   L. Seitz
Internet-Draft   RISE SICS
Intended status: Standards Track   G. Selander
Expires: November 22, 2018   Ericsson
                                                          E. Wahlstroem

                                                             S. Erdtman
                                                             Spotify AB
                                                          H. Tschofenig
                                                               ARM Ltd.
                                                           May 21, 2018

Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)

draft-ietf-ace-oauth-authz-12

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 CoAP, thus making a well-known and widely used authorization solution suitable for IoT devices. Existing specifications are used where possible, but where the constraints of IoT devices require it, extensions are added and profiles are defined.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Authorization is the process for granting approval to an entity to access a resource [RFC4949]. The authorization task itself can best be described as granting access to a requesting client, for a resource hosted on a device, the resource server (RS). This exchange is mediated by one or multiple authorization servers (AS). Managing authorization for a large number of devices and users 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 web applications on constrained nodes, this specification RECOMMENDS the use of CoAP [RFC7252] as replacement for HTTP.

A detailed treatment of constraints can be found in [RFC7228], and the different IoT deployments present a continuous range of device and network capabilities. Taking energy consumption as an example: At one end there are energy-harvesting or battery powered devices which have a tight power budget, on the other end there are mains-powered devices, and all levels in between.

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

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

More detailed, interoperable specifications can be found in profiles. Implementations may claim conformance with a specific profile, whereby implementations utilizing the same profile interoperate while implementations of different profiles are not expected to be interoperable. Some devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of low end devices. 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] and [I-D.ietf-ace-actors], such as client (C), resource server (RS), and authorization server (AS).

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as token and introspection at the AS and authz-info at the RS (see Section 5.8.1 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.

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

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).

We use the term "RS Information" for parameters describing characteristics of the RS (e.g. public key) that the AS provides to the client.

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, MQTT, BLE and QUIC.

A third building block is CBOR [RFC7049], for encodings where JSON [RFC8259] is not sufficiently compact. CBOR is a binary encoding designed for small code and message size, which may be used for encoding of self contained tokens, and also for encoding payload transferred in protocol messages.

A fourth building block is the compact CBOR-based secure message format COSE [RFC8152], which enables application layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC5246]). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which is an extension to the OAuth 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 [I-D.ietf-core-object-security].

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

Luckily, not every IoT device suffers from all constraints. The ACE framework nevertheless takes all these aspects into account and allows several different deployment variants to co-exist, rather than mandating a one-size-fits-all solution. 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:

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 if it needs to request additional information regarding a received access token. The RS makes a POST request to the introspection endpoint on the AS and receives information about the access token in the response. (See "Introspection" below.)

Access Tokens:

Access tokens are credentials needed to access protected resources. An access token is a data structure representing authorization permissions issued by the AS to the client. Access tokens are generated by the 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.

Proof of Possession Tokens:

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

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

The key bound to the access 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 encrypted and included in the access token. The PoP key is also encrypted for the client and sent together with the access token to the client.

Asymmetric PoP key:
An asymmetric key pair is generated on the client and the public key is sent to the AS (if it does not already have knowledge of the client’s public key). Information about the public key, which is the PoP key in this case, is either stored to be returned on introspection calls or included inside the access token and sent back to the requesting client. The RS can identify the client’s public key from the information in the token, which allows the client to use the corresponding private key for the proof of possession.

The access token is either a simple reference, or a structured information object (e.g., CWT [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 as data format, see Section 5, to request scopes and to be informed what scopes the access token actually authorizes.

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], an equivalent format using CBOR encoding (CWT) has been defined.

Introspection:

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

3.2. CoAP

CoAP is an application layer protocol similar to HTTP, but specifically designed for constrained environments. CoAP typically uses datagram-oriented transport, such as UDP, where reordering and loss of packets can occur. A security solution 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 1.2 [RFC6347] or TLS 1.2 [RFC5246]. 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 [I-D.ietf-core-object-security], which provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages. In OSCORE, the CoAP messages are wrapped in COSE objects and sent using CoAP.

This framework RECOMMENDS the use of CoAP as replacement for HTTP.

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 from an AS using the token endpoint and subsequently presents the access token to a 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.

The OAuth 2.0 framework defines a number of "protocol flows" via grant types, which have been extended further with extensions to OAuth 2.0 (such as RFC 7521 [RFC7521] and [I-D.ietf-oauth-device-flow]). What grant types works best depends on the usage scenario and RFC 7744 [RFC7744] describes many different IoT use cases but there are two preferred grant types, namely the Authorization Code Grant (described in Section 4.1 of [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.

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 broadcasted by a peripheral, including information about the primary services. In CoAP, as a second example, a client can make a request to "/.well-known/core" to obtain information about available resources, which are returned in a standardized format as described in [RFC6690].

```
   +--------+                               +---------------+
   |        |---(A)-- Token Request ------->|               |
   |        |                               | Authorization |
   |        |<--(B)-- Access Token ---------|    Server     |
   |        |       + RS Information        |               |
   |        |                               +---------------+
   |        |                                      ^ |
   |        |            Introspection Request  (D)| |
   | Client |                  (optional)          | |
   |        |                         Response     | |(E)
   |        |                         (optional)   | v
   |        |                               +--------------+
   |        |---(C)-- Token + Request ----->|              |
   |        |                               |   Resource   |
   |        |<--(F)-- Protected Resource ---|    Server    |
   |        |                               |              |
   +--------+                               +--------------+
```

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

Access Token Response (B):
If the AS successfully processes the request from the client, it returns an access token. It can also return additional parameters, referred to as "RS 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. More information about these parameters can be found in Section 5.6.4.

Resource Request (C):
The client interacts with the RS to request access to the protected resource and provides the access token. The protocol to use between the client and the RS is not restricted to CoAP. HTTP, HTTP/2, QUIC, MQTT, Bluetooth Low Energy, etc., are also viable candidates.

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

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

The Client and the RS mutually authenticate using the security protocol specified in the profile (see step B) and the keys obtained in the access token or the RS Information. The RS verifies that the token is integrity protected by the AS and compares the claims contained in the access token with the resource request. If the RS is online, validation can be handed over to the AS using token introspection (see messages D and E) over HTTP or CoAP.

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 5.7 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 used communication security protocol.

5. Framework

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

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

Proof-of-Possession
The ACE framework, by default, implements proof-of-possession for access tokens, i.e., that the token holder can prove being a holder of the key bound to the token. The binding is provided by the "cnf" claim [I-D.ietf-ace-cwt-proof-of-possession] 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 "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 data exchanged with the AS is encrypted and integrity protected. It is furthermore REQUIRED that the AS and the endpoint communicating with it (client or RS) perform mutual authentication.

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

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

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. If CoAP is used, this framework REQUIRES the use of CBOR [RFC7049] instead of JSON. Depending on the profile, the CBOR payload MAY be enclosed in a non-CBOR cryptographic wrapper.

5.1. Discovering Authorization Servers

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

Instead of the initial Unauthorized Resource Request message, C MAY look up the desired resource in a resource directory (cf. [I-D.ietf-core-resource-directory]).

5.1.1. Unauthorized Resource Request Message

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

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

Note: These conditions ensure that RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The authz-info endpoint MUST NOT be protected as specified above, in order to allow clients to upload access tokens to RS (cf. Section 5.8.1).

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Resource Server SHOULD provide proper AS Information to enable the Client to request an access token from RS’s AS as described in Section 5.1.2.

The handling of all client requests (including unauthorized ones) by the RS is described in Section 5.8.2.

5.1.2. AS Information

The AS Information is sent by RS as a response to an Unauthorized Resource Request message (see Section 5.1.1) to point the sender of the Unauthorized Resource Request message to RS’s AS. The AS information is a set of attributes containing an absolute URI (see Section 4.3 of [RFC3986]) that specifies the AS in charge of RS.

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

Figure 2 summarizes the parameters that may be part of the AS Information.

Name	CBOR Key	Value Type
AS	0	text string
nonce	5	byte string

Figure 2: AS Information parameters

Figure 3 shows an example for an AS Information message payload using CBOR [RFC7049] diagnostic notation, using the parameter names instead of the CBOR keys for better human readability.

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

Figure 3: AS Information payload example

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

Note: There is an ongoing discussion how freshness of access tokens

can be achieved in constrained environments. This specification for now assumes that RS and AS do not have a common understanding of time that allows RS to achieve its security objectives without explicitly adding a nonce.

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

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

Figure 4: AS Information example encoded in CBOR

5.2. Authorization Grants

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

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

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

For details on the different grant types, see the OAuth 2.0 framework [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.

5.3. Client Credentials

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

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

The OAuth framework [RFC6749] defines one client credential type, client id and client secret. [I-D.erdtman-ace-rpcc] adds raw-public-key and pre-shared-key to the client credentials types. Profiles of this framework MAY extend with additional client credentials client certificates.

5.4. AS Authentication

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

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

5.5. The Authorization Endpoint

The authorization endpoint is used to interact with the resource owner and obtain an authorization grant in certain grant flows. Since it requires the use of a user agent (i.e., browser), it is not expected that these types of grant flow will be used by constrained clients. This endpoint is therefore out of scope for this specification. Implementations should use the definition and recommendations of [RFC6749] and [RFC6819].

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

5.6. 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, however, 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, JSON or CBOR payloads may be supported. If JSON payloads are used, the semantics of Section 4 of the OAuth 2.0 specification MUST be followed (with additions as described below). If CBOR payload is supported, the semantics described below 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.

The default name of this endpoint in an url-path is ’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.

5.6.1. Client-to-AS Request

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

If CBOR is used then this parameter MUST be encoded as a CBOR map, where the "scope" parameter can additionally be formatted as a byte array, in order to allow compact encoding of complex scope structures.

When HTTP is used as a transport then the client makes a request to the token endpoint by sending the parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body, as defined in RFC 6749.

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

aud:
OPTIONAL. Specifies the audience for which the client is requesting an access token. If this parameter is missing, it is assumed that the client and the AS have a pre-established

understanding of the audience that an access token should address. If a client submits a request for an access token without specifying an "aud" parameter, and the AS does not have an implicit understanding of the "aud" value for this client, then the AS MUST respond with an error message using a response code equivalent to the CoAP response code 4.00 (Bad Request).

cnf:

OPTIONAL. This field contains information about the key the client would like to bind to the access token for proof-of-possession. It is RECOMMENDED that an AS reject a request containing a symmetric key value in the ’cnf’ field, since the AS is expected to be able to generate better symmetric keys than a potentially constrained client. See Section 5.6.4.5 for more details on the formatting of the ’cnf’ parameter.

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

Figure 5 shows a request for a token with a symmetric proof-of-possession key. Note that in this example it is assumed that transport layer communication security is used with a CBOR payload, therefore the Content-Type is "application/cbor". The content is displayed in CBOR diagnostic notation, without abbreviations for better readability.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Type: "application/cbor"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id" : "myclient",
  "aud" : "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 COSE is used to provide object-security, therefore the Content-Type is "application/cose".

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Type: "application/cose"
Payload:
  16(  # COSE_ENCRYPTED
      [ h’a1010a’, # protected header: {"alg" : "AES-CCM-16-64-128"}
      {5 : b64’ifUvZaHFgJM7UmGnjA’},  # unprotected header, IV
      b64’WXThuZo6TMCaZZqi6ef/8WHTjOdGk8kNzaIhIQ’ # ciphertext
      ]
  )


Decrypted payload:
{
  "grant_type" : "client_credentials",
  "client_id" : "myclient",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h’11’,
      "crv" : "P-256",
      "x" : b64’usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8’,
      "y" : b64’IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4’
    }
  }
}
```

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. Note that a transport layer based communication security profile with a CBOR payload is assumed in this example, therefore the Content-Type is "application/cbor". Also note that the client performs a password based authentication in this example by submitting its client_secret (see Section 2.3.1 of [RFC6749]).

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

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

5.6.2. 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 5.6.3.

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

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

profile:
OPTIONAL. This indicates the profile that the client MUST use towards the RS. See Section 5.6.4.4 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.

cnf:
REQUIRED if the token type is "pop" and a symmetric key is used. MUST NOT be present otherwise. This field contains the symmetric

proof-of-possession key the client is supposed to use. See Section 5.6.4.5 for details on the use of this parameter.

rs_cnf:
OPTIONAL if the token type is "pop" and asymmetric keys are used. MUST NOT be present otherwise. This field contains information about the public key used by the RS to authenticate. See Section 5.6.4.5 for details on the use of this parameter. If this parameter is absent, the AS assumes that the client already knows the public key of the RS.

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

Note that if CBOR Web Tokens [RFC8392] are used, the access token also contains a "cnf" claim [I-D.ietf-ace-cwt-proof-of-possession]. This claim is however consumed by a different party. The access token is created by the AS and processed by the RS (and opaque to the client) whereas the RS Information is created by the AS and processed by the client; it is never forwarded to the resource server.

Figure 8 summarizes the parameters that may be part of the RS Information.

Parameter name	Specified in
access_token	RFC 6749
token_type	RFC 6749
expires_in	RFC 6749
refresh_token	RFC 6749
scope	RFC 6749
state	RFC 6749
error	RFC 6749
error_description	RFC 6749
error_uri	RFC 6749
profile	[this document]
cnf	[this document]
rs_cnf	[this document]

Figure 8: RS Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key. Note that transport layer security with CBOR encoding is assumed in this example, therefore the Content-Type is "application/cbor".

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

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

5.6.3. Error Response

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

- The Content-Type MUST be specified by the communication security profile used between client and AS. When using CoAP the raw payload before being processed by the communication security protocol MUST be encoded as a CBOR map.
- 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
invalid_request	0
invalid_client	1
invalid_grant	2
unauthorized_client	3
unsupported_grant_type	4
invalid_scope	5
unsupported_pop_key	6

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" defined in Figure 10.

5.6.4. 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.

5.6.4.1. Audience

This parameter specifies for which audience the client is requesting a token. The formatting and semantics of these strings are application specific.

When encoded as a CBOR payload it is represented as a CBOR text string.

5.6.4.2. Grant Type

The abbreviations in Figure 11 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	RFC6749
authorization_code	1	RFC6749
client_credentials	2	RFC6749
refresh_token	3	RFC6749

Figure 11: CBOR abbreviations for common grant types

5.6.4.3. Token Type

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

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

The values in the "token_type" parameter MUST be CBOR text strings, if a CBOR encoding is used.

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

5.6.4.4. 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. Furthermore profiles MUST define 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 "profile" parameter. The textual representation of the profile identifier is just intended for human readability and MUST NOT be used in parameters and claims..

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

5.6.4.5. Confirmation

The "cnf" parameter identifies or provides the key used for proof-of-possession, while the "rs_cnf" parameter provides the raw public key of the RS. Both parameters use the same formatting and semantics as the "cnf" claim specified in [I-D.ietf-ace-cwt-proof-of-possession] when used with a CBOR encoding. When these parameters are used in JSON then the formatting and semantics of the "cnf" claim specified in RFC 7800 [RFC7800].

In addition to the use as a claim in a CWT, the "cnf" parameter is used in the following contexts with the following meaning:

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

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

Also note that the "rs_cnf" parameter is supposed to indicate the key that the RS uses to authenticate. If the access token is issued for an audience that includes several RS, this parameter MUST NOT be used, since it is them impossible to determine for which RS the key applies. This framework recommends to specify a different endpoint that the client can use to acquire RS authentication keys in such cases. The specification of such an endpoint is out of scope for this framework.

5.6.5. 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 Figure 12, using the given integer abbreviation for the map keys.

Note that we have aligned these abbreviations with the claim abbreviations defined in [RFC8392].

Name	CBOR Key	Value Type
aud	3	text string
client_id	8	text string
client_secret	9	byte string
response_type	10	text string
redirect_uri	11	text string
scope	12	text or byte string
state	13	text string
code	14	byte string
error	15	unsinged integer
error_description	16	text string
error_uri	17	text string
grant_type	18	unsigned integer
access_token	19	byte string
token_type	20	unsigned integer
expires_in	21	unsigned integer
username	22	text string
password	23	text string
refresh_token	24	byte string
cnf	25	map
profile	26	unsigned integer
rs_cnf	31	map

Figure 12: CBOR mappings used in token requests

5.7. The ’Introspect’ Endpoint

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

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

The default name of this endpoint in an url-path is ’introspect’, however implementations are not required to use this name and can define their own instead.

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

Note that supporting introspection is OPTIONAL for implementations of this framework.

5.7.1. RS-to-AS Request

The RS sends a POST request to the introspection endpoint at the AS, the profile MUST specify the Content-Type and wrapping of the payload. If CBOR is used, the payload MUST be encoded as a CBOR map with a "token" entry containing either the access token or a reference to the token (e.g., the cti). Further optional parameters representing additional context that is known by the RS to aid the AS in its response MAY be included.

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

For example, Figure 13 shows a 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 COSE is assumed in this example, therefore the Content-Type is "application/cose+cbor".

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

Figure 13: Example introspection request.

5.7.2. AS-to-RS 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 5.7.3.

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

cnf OPTIONAL. This field contains information about the proof-of-possession key that binds the client to the access token. See Section 5.6.4.5 for more details on the use of the "cnf" parameter.

profile OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.6.4.4 for more details on the formatting of this parameter.

rs_cnf OPTIONAL. If the RS has several keys it can use to authenticate towards the client, the AS can give the RS a hint using this parameter, as to which key it should use (e.g. if the AS previously informed the client about a public key the RS is holding). See Section 5.6.4.5 for more details on the use of this parameter.

For example, Figure 14 shows an AS response to the introspection request in Figure 13. Note that transport layer security is assumed in this example, therefore the Content-Type is "application/cbor".

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

Figure 14: Example introspection response.

5.7.3. Error Response

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

- If content is sent, the Content-Type MUST be set according to the specification of the communication security profile. If CoAP is used the payload MUST be encoded as a CBOR map.
- If the credentials used by 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 5.2 in RFC 6749 [RFC6749].
- If the RS 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 Figure 10.

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".

5.7.4. Mapping Introspection parameters to CBOR

If CBOR is used, the introspection request and response parameters MUST be mapped to CBOR types as specified in Figure 15, using the given integer abbreviation for the map key.

Note that we have aligned these abbreviations with the claim abbreviations defined in [RFC8392].

Parameter name	CBOR 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
client_id	8	text string
scope	12	text OR byte string
token_type	20	text string
username	22	text string
cnf	25	map
profile	26	unsigned integer
token	27	byte string
token_type_hint	28	text string
active	29	True or False
rs_cnf	30	map

Figure 15: CBOR Mappings to Token Introspection Parameters.

5.8. The Access Token

This framework RECOMMENDS the use of CBOR web token (CWT) as specified in [RFC8392].

In order to facilitate offline processing of access tokens, this draft uses the "cnf" claim from [I-D.ietf-ace-cwt-proof-of-possession] and specifies the "scope" claim for both JSON and CBOR web tokens.

The "scope" claim explicitly encodes the scope of a given access token. This claim follows the same encoding rules as defined in Section 3.3 of [RFC6749], but in addition implementers MAY use byte arrays as scope values, to achieve compact encoding of large scope elements. The meaning of a specific scope value is application specific and expected to be known to the RS running that application.

If the AS needs to convey a hint to the RS about which key it should use to authenticate towards the client, the rs_cnf claim MAY be used with the same syntax and semantics as defined in Section 5.6.4.5.

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 a "profile" claim in the access token, with the same syntax and semantics as defined in Section 5.6.4.4.

5.8.1. The ’Authorization Information’ Endpoint

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

This section defines a method for transporting the access token to the RS using 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 RS receiving the token MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with 2.01 (Created). This response MAY contain an identifier of the token (e.g., the cti for a CWT) as a payload, in order to allow the client to refer to the 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.

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

The RS MAY make an introspection request to validate the token before responding to the POST request to the authz-info endpoint.

Profiles MUST specify how the authz-info endpoint is protected, including how 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.

5.8.2. Client Requests to the RS

A RS receiving a client request MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession for that token.

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 not 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.

5.8.3. Token Expiration

Depending on the capabilities of the RS, there are various ways in which it can verify the validity of a received access token. 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 5.7. This requires the RS to have a reliable network connection to the AS and to be able to handle two secure sessions in parallel (C to RS and AS to RS).
- The RS and the AS both store a sequence number linked to their common security association. The AS increments this number for each access token it issues and includes it in the access token, which is a CWT. The RS keeps track of the most recently received sequence number, and only accepts tokens as valid, that are in a certain range around this number. This method does only require the RS to keep track of the sequence number. The method does not provide timely expiration, but it makes sure that older tokens cease to be valid after a certain number of newer ones got issued. For a constrained RS with no network connectivity and no means of reliably measuring time, this is the best that can be achieved.

If a token that authorizes a long running request such as 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.

6. Security Considerations

Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work, as well as the security considerations from [I-D.ietf-ace-actors]. Furthermore [RFC6819] provides additional security considerations for OAuth which apply to IoT deployments as well.

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest (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. 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 message needs to be publicly readable.

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

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client 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. Profiles MUST specify how confidentiality protection is provided, and additional protection can be applied by encrypting the token, for example encryption of CWTs is specified in Section 5.1 of [RFC8392].

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

Clients can at any time request a new proof-of-possession capable access token. If clients have that capability, the AS can keep the lifetime of the access token and the associated proof-of-possession key short and therefore 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 permissions. Furthermore access tokens using symmetric keys for proof-of-possession SHOULD NOT be targeted at an audience that contains more than one RS, since otherwise any RS in the audience that receives that access token can impersonate the client towards the other members of the audience.

6.1. Unprotected AS Information

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the AS information contained in an unprotected response from RS to an unauthorized request (c.f. Section 5.1.2) is authentic. It is therefore advisable to provide C with a (possibly hard-coded) list of trustworthy authorization servers. AS information responses referring to a URI not listed there would be ignored.

6.2. Use of Nonces for Replay Protection

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

6.3. Combining profiles

There may exist reasonable use cases where implementers want to combine different profiles of this framework, e.g., using an MQTT profile between client and RS, while using a DTLS profile for interactions between client and AS. Profiles should be designed in a way that the security of a protocol interaction does not depend on the specific security mechanisms used in other protocol interactions.

6.4. Error responses

The various error responses defined in this framework may leak information to an adversary. For example errors 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. 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.

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.

If access tokens are only integrity protected and not encrypted, they may reveal information to attackers listening on the wire, or able to acquire the access tokens in some other way. In the case of CWTs 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 (c.f. Section 5.1.2) may disclose information about RS and/or its existing relationship with C. It is advisable to include as little information as possible in an unencrypted response. 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

8.1. Authorization Server Information

This section establishes the IANA "ACE Authorization Server Information" registry. The registry has been created to use the "Expert Review Required" 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 grant type 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. OAuth Error Code CBOR Mappings Registry

This section establish the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" 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 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. 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 grant type abbreviation, 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.

8.3. OAuth Grant Type CBOR Mappings

This section establishes the IANA "OAuth Grant Type CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" 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 grant type as specified in Section 1.3 of [RFC6749].

CBOR Value CBOR abbreviation for this grant type. 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 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.

8.4. OAuth Access Token Types

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

- Name: "PoP"
- Change Controller: IETF
- Reference: [this document]

8.5. OAuth Token Type CBOR Mappings

This section eatables the IANA "Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" 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 token type as registered in the OAuth Access Token Types registry e.g., "Bearer".

CBOR Value CBOR abbreviation for this token type. 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 OAuth token type abbreviation, if one exists.

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

8.5.1. Initial Registry Contents

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

- Name: "pop"

- Value: 2
- Reference: [this document]
- Original Specification: [this document]

8.6. ACE Profile Registry

This section establishes the IANA "ACE Profile" registry. The registry has been created to use the "Expert Review Required" 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.

8.7. OAuth Parameter Registration

This section registers the following parameters in the "OAuth Parameters" registry [IANA.OAuthParameters]:

- Name: "aud"
- Parameter Usage Location: authorization request, token request
- Change Controller: IESG
- Reference: Section 5.6.1 of [this document]

- Name: "profile"
- Parameter Usage Location: token response
- Change Controller: IESG
- Reference: Section 5.6.4.4 of [this document]

- Name: "cnf"
- Parameter Usage Location: token request, token response
- Change Controller: IESG
- Reference: Section 5.6.4.5 of [this document]

- Name: "rs_cnf"
- Parameter Usage Location: token response
- Change Controller: IESG
- Reference: Section 5.6.4.5 of [this document]

8.8. OAuth CBOR Parameter Mappings Registry

This section establishes the IANA "Token Endpoint CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" 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 OAuth Parameter name, refers to the name in the OAuth parameter registry e.g., "client_id".

CBOR Key CBOR map key for this 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 allowable CBOR data types for values of this parameter.

Reference This contains a pointer to the public specification of the grant type abbreviation, 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.

Note that these mappings intentionally coincide with the CWT claim name mappings from [RFC8392].

8.9. OAuth Introspection Response Parameter Registration

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

- Name: "cnf"
- Description: Key to prove the right to use a PoP token.
- Change Controller: IESG
- Reference: Section 5.7.2 of [this document]

- Name: "profile"

- Description: The communication and communication security profile used between client and RS, as defined in ACE profiles.
- Change Controller: IESG
- Reference: Section 5.7.2 of [this document]

8.10. Introspection Endpoint CBOR Mappings Registry

This section establishes the IANA "Introspection Endpoint CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" 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 OAuth Parameter name, refers to the name in the OAuth parameter registry e.g., "client_id".

CBOR Key CBOR map key for this 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 allowable CBOR data types for values of this parameter.

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

This registry will be initially populated by the values in Figure 15. The Reference column for all of these entries will be this document.

8.11. 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: "scope"
- Claim Description: The scope of an access token as defined in [RFC6749].
- Change Controller: IESG
- Reference: Section 5.8 of [this document]

8.12. CBOR Web Token Claims

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

- o Claim Name: "scope"
- o Claim Description: The scope of an access token as defined in [RFC6749].
- o JWT Claim Name: N/A
- o Claim Key: 12
- o Claim Value Type(s): 0 (uint), 2 (byte string), 3 (text string)
- o Change Controller: IESG
- o Specification Document(s): Section 5.8 of [this document]

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 large parts of the security considerations where 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).

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

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

10. References

10.1. Normative References

[I-D.ietf-ace-cwt-proof-of-possession]
Jones, M., Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-02 (work in progress), March 2018.

[IANA.CborWebTokenClaims]
IANA, "CBOR Web Token (CWT) Claims", <https://www.iana.org/assignments/cwt/cwt.xhtml#claims-registry>.

[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.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>.

[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>.

[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>.

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

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

[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>.

10.2. Informative References

[I-D.erdtman-ace-rpcc] Seitz, L. and S. Erdtman, "Raw-Public-Key and Pre-Shared-Key as OAuth client credentials", draft-erdtman-ace-rpcc-02 (work in progress), October 2017.

[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-06 (work in progress), November 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-12 (work in progress), March 2018.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-13 (work in progress), March 2018.

[I-D.ietf-oauth-device-flow] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Flow for Browserless and Input Constrained Devices", draft-ietf-oauth-device-flow-09 (work in progress), April 2018.

[I-D.ietf-oauth-discovery]
Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", draft-ietf-oauth-discovery-10 (work in progress), March 2018.

[Margi10impact]
Margi, C., de Oliveira, B., de Sousa, G., Simplicio Jr, M., Barreto, P., Carvalho, T., 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), 2010 August.

[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>.

[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>.

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

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

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 e.g. AES) [Margi10impact]. 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 allows, but this may also require support for trusted third party assisted secret key establishment using transport or application layer security.

Small Amount of Memory:

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

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers 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 not limited to, other protocols such as HTTP, HTTP/2 or other specific 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:

This framework REQUIRES the use of CBOR [RFC7049] as data format. Where CBOR data needs to be protected, the use of COSE [RFC8152] is RECOMMENDED. Furthermore where self-contained tokens are needed, this framework RECOMMENDS the 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:

This framework RECOMMENDS the 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.

RS Information:

This framework defines the name "RS Information" for data concerning the RS that the AS returns to the client in an access token response (see Section 5.6.2). This includes the "profile" and the "rs_cnf" parameters. This aims at enabling scenarios, where a powerful client, supporting multiple profiles, needs to interact with a 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 [I-D.ietf-ace-cwt-proof-of-possession]. A semantically and syntactically identical request and response parameter is 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.

Auth-Info endpoint:

This framework introduces a new way of providing access tokens to a 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:

This framework RECOMMENDS the use of the client credentials grant 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 the (without manual intervention) obtain access tokens from the AS.

Introspection:

This framework RECOMMENDS the use of access token introspection 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 this framework RECOMMENDS the use of 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_types, scopes, and key types (symmetric/asymmetric) the RS supports. Also making it known to the AS which audience(s) the RS identifies itself with.
- Make sure that clients can discover the AS 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 [I-D.ietf-oauth-discovery]

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 RS Information (see step (B) of Figure 1).

    - Check that the RS Information provides the necessary security parameters (e.g., PoP key, information on communication security protocols supported by the RS).
- Send the token and request to the RS (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.
    + Verify that the token applies to this RS.
    + Check that the token has not expired (if the token provides expiration information).
    + Check the token's integrity.
    + Store the token so that it can be retrieved in the context of a matching request.
  * Process a request.

    + Set up communication security with the client.
    + Authenticate the client.
    + Match the client against existing tokens.
    + Check that tokens belonging to the client actually authorize the requested action.
    + Optionally: Check that the matching tokens are still valid, using introspection (if this is possible.)
  * Send a response following the agreed upon communication security.

Appendix C. Requirements on Profiles

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

- Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.6.4.4
- Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS over CoAP). This must provide encryption, integrity and replay protection. Section 5.6.4.4
- Specify how the client and the RS mutually authenticate. Section 4
- Specify the Content-format of the protocol messages (e.g., "application/cbor" or "application/cose+cbor"). Section 4
- Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol. Section 5.6.4.3
- Specify a unique profile identifier. Section 5.6.4.4
- If introspection is supported: Specify the communication and security protocol for introspection.Section 5.7
- Specify the communication and security protocol for interactions between client and AS. Section 5.6
- Specify how/if the authz-info endpoint is protected, including how error responses are protected. Section 5.8.1
- Optionally define other methods of token transport than the authz-info endpoint. Section 5.8.1

Appendix D. Assumptions on AS knowledge about C and RS

This section lists the assumptions on what an AS should know about a client and a RS in order to be able to respond to requests to the token and 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).
- The expiration time for access tokens issued to this RS (unless the RS accepts a default time chosen by the AS).
- The symmetric key shared between client or RS and AS (if any).
- The raw public key of the client or RS (if any).

Appendix E. Deployment Examples

There is a large variety of IoT deployments, as is indicated in Appendix A, and this section highlights a few common variants. This section is not normative but illustrates how the framework can be applied.

For each of the deployment variants, there are a number of possible security setups between clients, resource servers and authorization servers. The main focus in the following subsections is on how authorization of a client request for a resource hosted by a RS is performed. This requires the security of the requests and responses between the clients and the RS to consider.

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

E.1. Local Token Validation

In this scenario, 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 16.

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

```
         Authorization
  Client    Server
    |         |
    |<=======>| DTLS Connection Establishment
    |         |   to identify the AS
    |         |
A:  +-------->| Header: POST (Code=0.02)
    |  POST   | Uri-Path:"token"
    |         | Content-Type: application/cbor
    |         | Payload: <Request-Payload>
    |         |
B:  |<--------+ Header: 2.05 Content
    |  2.05   | Content-Type: application/cbor
    |         | Payload: <Response-Payload>
    |         |
```

Figure 16: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 17. Note that a transport layer security based communication security profile is used in this example, therefore the Content-Type is "application/cbor".

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

Response-Payload :
{
  "access_token" : b64'SlAV32hkKG ...',
  "token_type" : "pop",
  "csp" : "DTLS",
  "rs_cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x"   : b64'MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y"   : b64'4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
    }
  }
}
```

Figure 17: Request and Response Payload Details.

The content of the access token is shown in Figure 18.

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

Figure 18: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 19 - Figure 20.

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

```
          Resource
 Client     Server
    |         |
C:  +-------->| Header: POST (Code=0.02)
    |  POST   | Uri-Path:"authz-info"
    |         | Payload: SlAV32hkKG ...
    |         |
    |<--------+ Header: 2.04 Changed
    |  2.04   |
    |         |
```

Figure 19: Access Token provisioning to RS

The client and the RS runs the DTLS handshake using the raw public keys established in step B and C.

The client sends the CoAP request GET to /temperature on RS over DTLS. The RS verifies that the request is authorized, based on previously established security context.
F: The RS responds with a resource representation over DTLS.

```
             Resource
  Client     Server
     |          |
     |<=======>| DTLS Connection Establishment
     |          |   using Raw Public Keys
     |          |
     +--------->| Header: GET (Code=0.01)
     | GET      | Uri-Path: "temperature"
     |          |
     |          |
     |          |
 F:  |<---------+ Header: 2.05 Content
     | 2.05     | Payload: <sensor value>
     |          |
```

Figure 20: Resource Request and Response protected by DTLS.

E.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), a 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 21.

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

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

A: The client sends the request using POST to the token endpoint at AS. The request contains the Audience parameter set to "PACS1337" (PACS, Physical Access System), a value the that the online door in question identifies itself with. The AS generates an access token as an opaque string, which it can match to the specific client, a targeted audience and a symmetric key. The security is provided by identifying the AS on transport layer using a pre shared security context (psk, rpk or certificate) and then the client is identified using client_id and client_secret as in classic OAuth.
B: The AS responds with the an access token and RS Information, the latter containing a symmetric key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key is used as the PreSharedKey.

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

Figure 21: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 22.

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

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

Figure 22: Request and Response Payload for C offline

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

C: Next, the client POSTs the access token to the authz-info endpoint in the RS. This is a plain CoAP request, i.e., no DTLS between client and RS. Since the token is an opaque string, the RS cannot verify it on its own, and thus defers to respond the client with a status code until after step E.

D: The RS forwards the token to the introspection endpoint on the AS. Introspection assumes a secure connection between the AS and the RS, e.g., using transport of application layer security. In the example AS is identified using pre shared security context (psk, rpk or certificate) while RS is acting as client and is identified with client_id and client_secret.

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

After receiving message E, the RS responds to the client’s POST in step C with the CoAP response code 2.01 (Created).

```
              Resource
     Client    Server
       |         |
   C:  +-------->| Header: POST (T=CON, Code=0.02)
       |  POST   | Uri-Path:"authz-info"
       |         | Content-Type: "application/cbor"
       |         | Payload: b64'SlAV32hkKG ...''
       |         |
       |         |     Authorization
       |         |       Server
       |         |          |
       |      D: +--------->| Header: POST (Code=0.02)
       |         |  POST    | Uri-Path: "introspect"
       |         |          | Content-Type: "application/cbor"
       |         |          | Payload: <Request-Payload>
       |         |          |
       |      E: |<---------+ Header: 2.05 Content
       |         |  2.05    | Content-Type: "application/cbor"
       |         |          | Payload: <Response-Payload>
       |         |          |
       |         |
       |<--------+ Header: 2.01 Created
       |  2.01   |
       |         |
```

Figure 23: Token Introspection for C offline

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

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

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

Figure 24: 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.

```
              Resource
     Client    Server
       |         |
       |<=======>| DTLS Connection Establishment
       |         |   using Pre Shared Key
       |         |
       +-------->| Header: PUT (Code=0.03)
       | PUT     | Uri-Path: "state"
       |         | Payload: <new state for the lock>
       |         |
   F:  |<--------+ Header: 2.04 Changed
       | 2.04    | Payload: <new state for the lock>
       |         |
```

Figure 25: Resource request and response protected by OSCORE

Appendix F. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

F.1. Version -11 to -12

- Moved the Request error handling to a section of its own.
- Require the use of the abbreviation for profile identifiers.
- Added rs_cnf parameter in the introspection response, to inform RS’ with several RPKs on which key to use.
- Allowed use of rs_cnf as claim in the access token in order to inform an RS with several RPKs on which key to use.
- Clarified that profiles must specify if/how error responses are protected.
- Fixed label number range to align with COSE/CWT.
- Clarified the requirements language in order to allow profiles to specify other payload formats than CBOR if they do not use CoAP.

F.2. Version -10 to -11

- Fixed some CBOR data type errors.
- Updated boilerplate text

F.3. Version -09 to -10

- Removed CBOR major type numbers.
- Removed the client token design.
- Rephrased to clarify that other protocols than CoAP can be used.
- Clarifications regarding the use of HTTP

F.4. Version -08 to -09

- Allowed scope to be byte arrays.
- Defined default names for endpoints.
- Refactored the IANA section for briefness and consistency.
- Refactored tables that define IANA registry contents for consistency.
- Created IANA registry for CBOR mappings of error codes, grant types and Authorization Server Information.
- Added references to other document sections defining IANA entries in the IANA section.

F.5. Version -07 to -08

- Moved AS discovery from the DTLS profile to the framework, see Section 5.1.
- Made the use of CBOR mandatory. If you use JSON you can use vanilla OAuth.
- Made it mandatory for profiles to specify C-AS security and RS-AS security (the latter only if introspection is supported).
- Made the use of CBOR abbreviations mandatory.
- Added text to clarify the use of token references as an alternative to CWTs.
- Added text to clarify that introspection must not be delayed, in case the RS has to return a client token.
- Added security considerations about leakage through unprotected AS discovery information, combining profiles and leakage through error responses.
- Added privacy considerations about leakage through unprotected AS discovery.
- Added text that clarifies that introspection is optional.
- Made profile parameter optional since it can be implicit.
- Clarified that CoAP is not mandatory and other protocols can be used.
- Clarified the design justification for specific features of the framework in appendix A.
- Clarified appendix E.2.
- Removed specification of the "cnf" claim for CBOR/COSE, and replaced with references to [I-D.ietf-ace-cwt-proof-of-possession]

F.6. Version -06 to -07

- Various clarifications added.
- Fixed erroneous author email.

F.7. Version -05 to -06

- Moved sections that define the ACE framework into a subsection of the framework Section 5.
- Split section on client credentials and grant into two separate sections, Section 5.2, and Section 5.3.
- Added Section 5.4 on AS authentication.
- Added Section 5.5 on the Authorization endpoint.

F.8. Version -04 to -05

- Added RFC 2119 language to the specification of the required behavior of profile specifications.
- Added Section 5.3 on the relation to the OAuth2 grant types.
- Added CBOR abbreviations for error and the error codes defined in OAuth2.
- Added clarification about token expiration and long-running requests in Section 5.8.3
- Added security considerations about tokens with symmetric pop keys valid for more than one RS.
- Added privacy considerations section.
- Added IANA registry mapping the confirmation types from RFC 7800 to equivalent COSE types.
- Added appendix D, describing assumptions about what the AS knows about the client and the RS.

F.9. Version -03 to -04

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

F.10. Version -02 to -03

- Removed references to draft-ietf-oauth-pop-key-distribution since the status of this draft is unclear.
- Copied and adapted security considerations from draft-ietf-oauth-pop-key-distribution.
- Renamed "client information" to "RS information" since it is information about the RS.
- Clarified the requirements on profiles of this framework.

- Clarified the token endpoint protocol and removed negotiation of "profile" and "alg" (section 6).
- Renumbered the abbreviations for claims and parameters to get a consistent numbering across different endpoints.
- Clarified the introspection endpoint.
- Renamed token, introspection and authz-info to "endpoint" instead of "resource" to mirror the OAuth 2.0 terminology.
- Updated the examples in the appendices.

F.11. Version -01 to -02

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

F.12. Version -00 to -01

- Changed 5.1. from "Communication Security Protocol" to "Client Information".
- Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP access token request profile for IoT.

  * Allow the client to indicate preferences for the communication security protocol.
  * Defined the term "Client Information" for the additional information returned to the client in addition to the access token.
  * Require that the messages between AS and client are secured, either with (D)TLS or with COSE_Encrypted wrappers.
  * Removed dependency on OSCOAP and added generic text about object security instead.
  * Defined the "rpk" parameter in the client information to transmit the raw public key of the RS from AS to client.
  * (D)TLS MUST use the PoP key in the handshake (either as PSK or as client RPK with client authentication).
  * Defined the use of x5c, x5t and x5tS256 parameters when a client certificate is used for proof of possession.

  - Defined "tktn" parameter for signaling for how to transfer the access token.

- Added 5.2. the CoAP Access-Token option for transferring access tokens in messages that do not have payload.
- 5.3.2. Defined success and error responses from the RS when receiving an access token.
- 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
- Appendix B Added list of roles and responsibilities for C, AS and RS.

Authors' Addresses

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

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

Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2018

M. Jones
Microsoft
L. Seitz
RISE SICS
G. Selander
Ericsson AB
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
ARM Ltd.
June 30, 2017

Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)
draft-jones-ace-cwt-proof-of-possession-01

Abstract

This specification describes how to declare in a CBOR Web Token (CWT) 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 the presenter being a holder-of-key. This specification provides equivalent functionality to "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" (RFC 7800), but using CBOR and CWTs rather than JSON and JWTs.

Status of This Memo

This Internet-Draft will expire on January 1, 2018.

Copyright Notice

Table of Contents

1. Introduction

This specification describes how a CBOR Web Token [CWT] 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 the presenter being a holder-of-key. This specification provides equivalent functionality to "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" [RFC7800], but using CBOR [RFC7049] and CWTs [CWT] rather than JSON [RFC7159] and JWTs [JWT].

1.1. Notational Conventions

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

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

2. Terminology

This specification uses terms defined in the CBOR Web Token [CWT], [I-D.ietf-cose-msg], and Concise Binary Object Representation (CBOR) [RFC7049] specifications.

These terms are defined by this specification:

Issuer
Party that creates the CWT and binds the proof-of-possession key to it.

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

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

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 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. If the CWT contains a "sub" (subject) claim [CWT], the presenter is normally the subject identified by the CWT. (In some applications, the subject identifier will be relative to the issuer identified by the "iss" (issuer) claim [CWT].) If the CWT contains no "sub" claim, the presenter is normally the issuer identified by the CWT using the "iss" claim. 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. At least one of the "sub" and "iss" claims is typically present in the CWT and some use cases may require that both be present.

3.1. Confirmation Claim

The "cnf" claim is used in the CWT to contain members used to identify the proof-of-possession key. Other members of the "cnf" map may be defined because a proof-of-possession key may not be the only means of confirming the authenticity of the token. This 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 6.2 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; thus, at most one of the "COSE_Key" and "Encrypted_COSE_Key" confirmation values defined below 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].

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 [I-D.ietf-cose-msg] representing the corresponding asymmetric public key. The following example (using JSON notation) demonstrates such a declaration in the CWT Claims Set of a CWT:

```
{
 "iss": "https://server.example.com",
 "aud": "https://client.example.org",
 "exp": 1361398824,
 "cnf":{
   "COSE_Key":{
     "kty": "EC",
     "crv": "P-256",
     "x": "18wHLeIgW9wVN6VD1Txgpqy2LszYkMf6J8njVAibvhM",
     "y": "-V4dS4UaLMgP_4fY4j8ir7cl1TXlFdAgcx55o7TkcSA"
    }
  }
}
```

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 [CWT]. If the CWT is not encrypted, the symmetric key MUST be encrypted as described below.

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 [I-D.ietf-cose-msg] representing the symmetric key encrypted to a key known to the recipient using COSE_Encrypt or COSE_Encrypt0.

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

```
{
 "kty": "oct",
 "alg": "HS256",
 "k": "ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE"
}
```

The COSE_Key representation is used as the plaintext when encrypting the key. The COSE_Key could, for instance, be encrypted using a COSE_Encrypt0 representation using the AES-CCM-16-64-128 algorithm.

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

```
  {
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "exp": 1311281970,
   "iat": 1311280970,
   "cnf":{
     "Encrypted_COSE_Key":
       "(TBD)"
     }
  }
```

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

The proof-of-possession key can also be identified by the use of 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 (using JSON notation) demonstrates such a declaration in the CWT Claims Set of a CWT:

```
  {
   "iss": "https://server.example.com",
   "aud": "https://client.example.org",
   "exp": 1361398824,
   "cnf":{
     "kid": "dfd1aa97-6d8d-4575-a0fe-34b96de2bfad"
    }
  }
```

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.

3.5. Specifics Intentionally Not Specified

Proof of possession is typically 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".

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 another means of proving possession of the key when it is a symmetric key is to encrypt the key to the recipient. The means of obtaining a key for the recipient is likewise protocol specific.

4. Security Considerations

All of the security considerations that are discussed in [CWT] 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 4.1.3 of [JWT], as it provides different protections. Proof of possession can be used by recipients to reject messages from unauthorized senders. Audience restriction can be used by recipients to reject messages intended for different recipients.

A recipient might not understand the "cnf" claim. Applications that require the proof-of-possession keys communicated with it to be understood and processed must ensure that the parts of this specification that they use are implemented.

Proof of possession via encrypted symmetric secrets is subject to replay attacks. This attack can, for example, be avoided when a signed nonce or challenge is used since the recipient can use a distinct nonce or challenge for each interaction. Replay can also be avoided if a sub-key is derived from a shared secret that is specific to the instance of the PoP demonstration.

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.

5. Privacy Considerations

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

6. IANA Considerations

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

Values are registered on a Specification Required [RFC5226] 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 to the IESG’s attention (using the iesg@ietf.org mailing list) for resolution.

Criteria that should be applied by the Designated Experts include determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and evaluating the security properties of the item being registered and whether the registration makes sense.

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.

6.1. CBOR Web Token Claims Registration

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

6.1.1. Registry Contents

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

6.2. 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.

6.2.1. 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. 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.

6.2.2. Initial Registry Contents

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

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

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

7. References

7.1. Normative References

[CWT] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", Work in Progress, draft-ietf-ace-cbor-web-token-06, June 2017, <https://tools.ietf.org/html/draft-ietf-ace-cbor-web-token-06>.

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

[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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

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

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

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

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <http://www.rfc-editor.org/info/rfc6125>.

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

7.2. Informative References

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

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7159, 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/>.

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

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

Acknowledgements

Thanks to the following people for their reviews of the specification: Michael Richardson and Jim Schaad.

Open Issues

- Convert the examples from JSON/JWT to CBOR/CWT.

Document History

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

-01

- Tracked CBOR Web Token (CWT) Claims Registry updates.
- Addressed review comments by Michael Richardson and Jim Schaad.
- Added co-authors.

-00

- Created the initial draft from RFC 7800.

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

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

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

Phone: +46702691499
Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 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 will expire on April 28, 2018.

Copyright Notice

Table of Contents

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:

- a master secret

- the sender identifier

- 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].

- an AEAD algorithm

- a KDF algorithm

- a salt

- 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'+a+Dg2jjU+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'+a+Dg2jjU+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].

- (Optional) discovery process of how the client finds the right AS for an RS it wants to send a request to: Not specified

- communication protocol the client and the RS must use: CoAP

- security protocol the client and RS must use: OSCORE

- how the client and the RS mutually authenticate: Implicitly by possession of a common OSCORE security context

- Content-format of the protocol messages: "application/cose+cbor"

- proof-of-possession protocol(s) and how to select one; which key types (e.g. symmetric/asymmetric) supported: OSCORE algorithms; pre-established symmetric keys

- profile identifier: coap_oscore

- (Optional) how the RS talks to the AS for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE)

- how the client talks to the AS for requesting a token: HTTP/CoAP (+ TLS/DTLS/OSCORE)

- how/if the /authz-info endpoint is protected: Security protocol above

- (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:

- a symmetric or public key (associated to the RS)

- 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'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
        "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+1rb7btKLv8EX4'
      }
    }
   }
```

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'cGJ90UiglWiGahtagnv8usWxHK2PmfnHKwXPS54m0kT',
           "y" : b64'reASjpkttcsz+1rb7btKLv8EX4IBOL+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’+a+Dg2jjU+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’+a+Dg2jjU+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).

- 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).

- 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]).

```
                    Resource
           Client    Server
           |          |
           |          |
           +--------->| Header: POST (Code=0.02)
           | POST     | Uri-Path:"authz-info"
           |          | Content-Type: application/cbor
           |          | Payload: access token
           |          |
           |          |
           +--------->| Header: POST (Code=0.02)
           |   POST   | Uri-Path: "/.well-known/edhoc"
           |          | Content-Type: application/edhoc
           |          | Payload: EDHOC message_1
           |          |
           |<---------+ Header: 2.04 Changed
           |   2.04   | Content-Type: application/edhoc
           |          | Payload: EDHOC message_2
           |          |
           +--------->| Header: POST (Code=0.02)
           |   POST   | Uri-Path: "/.well-known/edhoc"
           |          | Content-Type: application/edhoc
           |          | Payload: EDHOC message_3
           |          |
           |<---------+ Header: 2.04 Changed
           |   2.04   |
           |          |
    start of protected communication
           |          |
           +--------->| CoAP request +
           |  OSCORE  | Object-Security option
           | request  |
           |          |
           |<---------+ CoAP response +
           |  OSCORE  | Object-Security option
           | response |
           |          |
```

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   G. Selander
Internet-Draft   J. Mattsson
Intended status: Standards Track   F. Palombini
Expires: January 4, 2018   Ericsson AB
                                                           July 03, 2017

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-07

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56a [SP-800-56a], and HKDF [RFC5869]. CBOR [RFC7049] and COSE [I-D.ietf-cose-msg] are used to implement these standards.

1.1. Terminology

This document use the same informational CBOR Data Definition Language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl] grammar as COSE (see Section 1.3 of [I-D.ietf-cose-msg]). A vertical bar | denotes byte string concatenation.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

2. Protocol Overview

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and TLS 1.3, EDHOC is built on a variant of the SIGMA protocol which provide identity protection, and like TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an AEAD algorithm is shown in Figure 1.

```
      Party U                                                 Party V
         |                          E_U                          |
         +------------------------------------------------------>|
         |                                                       |
         |         E_V, Enc(K_2; ID_V, Sig(V; E_U, E_V);)        |
         |<------------------------------------------------------+
         |                                                       |
         |           Enc(K_3; ID_U, Sig(U; E_V, E_U);)           |
         +------------------------------------------------------>|
         |                                                       |
```

Figure 1: AEAD variant of the SIGMA-I protocol

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive the keying material. The messages are signed, MACed, and encrypted.

o E_U and E_V are the ECDH ephemeral public keys of U and V, respectively.

o ID_U and ID_V are identifiers for the public keys of U and V, respectively.

o Sig(U; .) and S(V; .) denote signatures made with the private key of U and V, respectively.

o Enc(K; P; A) denotes AEAD encryption of plaintext P and additional authenticated data A using the key K derived from the shared secret. The AEAD MUST NOT be replaced by plain encryption, see Section 8.

As described in Appendix B of [SIGMA], in order to create a "full-fledged" protocol some additional protocol elements are needed. EDHOC adds:

o Explicit session identifiers S_U, S_V different from other concurrent session identifiers (EDHOC or other used protocol identifier) chosen by U and V, respectively.

o Explicit nonces N_U, N_V chosen freshly and anew with each session by U and V, respectively.

o Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.

EDHOC also makes the following additions:

- o Negotiation of key derivation, encryption, and signature algorithms:
  - * U proposes one or more algorithms of the following kinds:
    - + HKDF
    - + AEAD
    - + Signature verification
    - + Signature generation
  - * V selects one algorithm of each kind
- o Verification of common preferred ECDH curve:
  - * U lists supported ECDH curves in order of preference
  - * V verifies that the ECDH curve of the ephemeral key is the most preferred common curve
- o Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR and COSE libraries. EDHOC does not put any requirement on the lower layers and can therefore be also be used e.g. in environments without IP.

This paper is organized as follows: Section 3 specifies general properties of EDHOC, including formatting of the ephemeral public keys and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, and Appendix A provides a wealth of test vectors to ease implementation and ensure interoperability.

3. EDHOC Overview

EDHOC consists of three messages (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consists of a CBOR array where the first element is an int specifying the message type (MSG_TYPE). After creating EDHOC message_3, Party U can derive the traffic key (master secret) and protected application data can therefore be sent in parallel with EDHOC message_3. The application data may be protected using the negotiated AEAD algorithm and the explicit session identifiers S_U and S_V. EDHOC may be used with the media type application/edhoc defined in Section 7.

```
      Party U                                                 Party V
         |                                                       |
         | ------------------ EDHOC message_1 -----------------> |
         |                                                       |
         | <----------------- EDHOC message_2 ------------------ |
         |                                                       |
         | ------------------ EDHOC message_3 -----------------> |
         |                                                       |
         | <----------- Protected Application Data ------------> |
         |                                                       |
```

Figure 2: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or certificates (Cert). EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed.

EDHOC also allows opaque application data (APP_1, APP_2, APP_3) to be sent in the respective messages. APP_1 is unprotected, APP_2 is protected (encrypted and integrity protected), and APP_3 is protected and mutually authenticated. When EDHOC is used with asymmetric key authentication APP_2 is sent to an unauthenticated party, but with symmetric key authentication APP_2 is mutually authenticated.

3.1. Formatting of the Ephemeral Public Keys

The ECDH ephemeral public key SHALL be formatted as a COSE_Key of type EC2 or OKP according to section 13.1 and 13.2 of [I-D.ietf-cose-msg]. The curve X25519 is mandatory to implement. For Elliptic Curve Keys of type EC2, compact representation and compact output as per [RFC6090] SHALL be used, i.e. the ’y’ parameter SHALL NOT be present in the The COSE_Key object. COSE [I-D.ietf-cose-msg] always use compact output for Elliptic Curve Keys of type EC2.

3.2. Key Derivation

Key and IV derivation SHALL be done as specified in Section 11.1 of [I-D.ietf-cose-msg] with the following input:

- The PRF SHALL be the HKDF [RFC5869] in the ECDH-SS w/ HKDF negotiated during the message exchange (HKDF_V).

- The secret SHALL be the ECDH shared secret as defined in Section 12.4.1 of [I-D.ietf-cose-msg].

- The salt SHALL be the PSK when EDHOC is authenticated with symmetric keys and nil when EDHOC is authenticated with asymmetric keys.

- The fields in the context information COSE_KDF_Context SHALL have the following values:

  - AlgorithmID is a tstr as defined below

  - PartyUInfo = PartyVInfo = (nil, nil, nil)

  - keyDataLength is a uint as defined below

  - protected SHALL be a zero length bstr

  - other is a bstr SHALL be aad_2, aad_3, or exchange_hash

where exchange_hash, in non-CDDL notation, is:

```
exchange_hash = H( H( message_1 | message_2 ) | message_3 )
```

where H() is the hash function in HKDF_V.

For message_i the key, called K_i, SHALL be derived using other = aad_i, where i = 2 or 3. The key SHALL be derived using AlgorithmID set to the name of the negotiated AEAD (AEAD_V), and keyDataLength equal to the key length of AEAD_V.

If the AEAD algorithm requires an IV, then IV_i for message_i SHALL be derived using other = aad_i, where i = 2 or 3. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in section 12.1.2. of [I-D.ietf-cose-msg], and keyDataLength equal to the IV length of AEAD_V.

Application specific traffic keys and other data SHALL be derived using other = exchange_hash. AlgorithmID SHALL be a tstr defined by the application and SHALL be different for different data being derived (an example is given in Appendix C.2). keyDataLength is set to the length of the data being derived.

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

EDHOC supports authentication with raw public keys (RPK) and certificates (Cert) with the requirements that:

- Party U SHALL be able to identify Party V’s public key using ID_V.

- Party V SHALL be able to identify Party U’s public key using ID_U.

ID_U and ID_V SHALL either contain the credential used for authentication (e.g. x5bag or x5chain) or uniquely identify the credential used for authentication (e.g. x5t), see [I-D.schaad-cose-x509]. Party U and V MAY retrieve the other party’s credential out of band. Optionally, ID_U and ID_V are complemented with the additional parameters HINT_ID_U and HINT_ID_V containing information about how to retrieve the credential of Party U and Party V, respectively (e.g. x5u), see [I-D.schaad-cose-x509].

Party U and Party V MAY use different type of credentials, e.g. one uses RPK and the other uses Cert. Party U and Party V MAY use different signature algorithms.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

```
Party U                                                          Party V
|                      S_U, N_U, E_U, ALG_1, APP_1                     |
+--------------------------------------------------------------------->|
|                               message_1                              |
|                                                                      |
|S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; Sig(V; ID_V, aad_2, APP_2); aad_2)|
|<---------------------------------------------------------------------+
|                               message_2                              |
|                                                                      |
|           S_V, Enc(K_3; Sig(U; ID_U, aad_3, APP_3); aad_3)           |
+--------------------------------------------------------------------->|
|                               message_3                              |
```

Figure 3: EDHOC with asymmetric key authentication.

4.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with asymmetric keys, the COSE algorithms ECDH-SS + HKDF-256, AES-CCM-64-64-128, and EdDSA are mandatory to implement.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  SIGs_V : alg_array,
  SIGs_U : alg_array,
  ? APP_1 : bstr
]

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 1

- S_U - variable length session identifier

- N_U - 64-bit random nonce

- E_U - the ephemeral public key of Party U

- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference

- HKDFs_U - supported ECDH-SS w/ HKDF algorithms

- AEADs_U - supported AEAD algorithms

- SIGs_V - signature algorithms, with which Party U supports verification

- SIGs_U - signature algorithms, with which Party U supports signing

- APP_1 - bstr containing opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_U.

- Choose a session identifier S_U which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other concurrent session identifiers used by Party U. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_U SHALL be different from the concurrently used session identifiers of that protocol.

- Format message_1 as specified in Section 4.2.1.

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.

- Verify that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- For EC2 curves, validate that E_U is a valid point by verifying that there is a solution to the curve definition for the given parameter ’x’.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol

MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message MUST include the following diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U:

```
ERR_MSG = "Curve not supported; X"
```

where X is the first curve in ECDH-Curves_U that V supports, encoded as in Table 22 of {{I-D.ietf-cose-msg}}.

- Pass APP_1 to the application.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr,
  SIG_V : int / tstr,
  SIG_U : int / tstr
)

aad_2 : bstr
```

where aad_2, in non-CDDL notation, is:

```
aad_2 = H( message_1 | [ data_2 ] )
```

where:

- MSG_TYPE = 2

- S_V - variable length session identifier

- N_V - 64-bit random nonce

- E_V - the ephemeral public key of Party V
- HKDF_V - a single chosen algorithm from HKDFs_U
- AEAD_V - a single chosen algorithm from AEADs_U
- SIG_V - a single chosen algorithm from SIGs_V with which Party V signs
- SIG_U - a single chosen algorithm from SIGs_U with which Party U signs
- COSE_ENC_2 has the following fields and values:
  - external_aad = aad_2
  - plaintext = [COSE_SIG_V, ? APP_2]
- COSE_SIG_V is a COSE_Sign1 object with the following fields and values:
  - protected = { abc : ID_V, ? xyz : HINT_ID_V }
  - detached payload = aad_2, ? APP_2
- abc - any COSE map label that can identify a public key, see Section 4.1
- ID_V - identifier for the public key of Party V
- xyz - any COSE map label for information about how to retrieve the credential of Party V, see Section 4.1
- HINT_ID_V - information about how to retrieve the credential of Party V
- APP_2 - bstr containing opaque application data
- H() - the hash function in HKDF_V

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_V.

- Choose a session identifier S_V which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party V. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_V SHALL be different from the concurrently used session identifiers of that protocol.

- Select HKDF_V, AEAD_V, SIG_V, and SIG_U from the algorithms proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- Format message_2 as specified in Section 4.3.1:

  * COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the private key of Party V.

  * COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- Verify that HKDF_V, AEAD_V, SIG_V, and SIG_U were proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- Verify (OPTIONAL) that N_V has not been received before.

- For EC2 curves, validate that E_V is a valid point by verifying that there is a solution to the curve definition for the given parameter ’x’.

- Verify message_2 as specified in Section 4.3.1:

  * COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

  * COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the public key of Party V.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- o Pass APP_2 to the application.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 : bstr
```

where aad_3, in non-CDDL notation, is:

```
aad_3 = H( H( message_1 | message_2 ) | [ data_3 ] )
```

where:

- o MSG_TYPE = 3

- o COSE_ENC_3 has the following fields and values:

  - * external_aad = aad_3

  - * plaintext = [COSE_SIG_U, ? APP_3]

- o COSE_SIG_U is a COSE_Sign1 object with the following fields and values:

  - * protected = { abc : ID_U, ? xyz : HINT_ID_U }

  - * detached payload = aad_3, ? APP_3

- o abc - any COSE map label that can identify a public key, see Section 4.1

- ID_U - identifier for the public key of Party U

- xyz - any COSE map label for information about how to retrieve the credential of Party U, see Section 4.1

- HINT_ID_V - information about how to retrieve the credential of Party U

- APP_3 - bstr containing opaque application data

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 4.4.1:

  * COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the private key of Party U.

  * COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.

- Verify message_3 as specified in Section 4.4.1:

  * COSE_Encrypt0 is decrypted as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

  * COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the public key of Party U.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_3 to the application.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

EDHOC supports authentication with pre-shared keys. Party U and V are assumed to have a pre-shared uniformly random key (PSK) with the requirement that:

- Party V SHALL be able to identify the PSK using KID.

KID may optionally contain information about how to retrieve the PSK.

EDHOC with symmetric key authentication is illustrated in Figure 4.

```
Party U                                                       Party V
|                S_U, N_U, E_U, ALG_1, KID, APP_1                   |
+------------------------------------------------------------------>|
|                             message_1                             |
|                                                                   |
|         S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; APP_2; aad_2)         |
|<------------------------------------------------------------------+
|                             message_2                             |
|                                                                   |
|                    S_V, Enc(K_3; APP_3; aad_3)                    |
+------------------------------------------------------------------>|
|                             message_3                             |
```

Figure 4: EDHOC with symmetric key authentication.

5.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with symmetric keys, the COSE algorithms ECDH-SS + HKDF-256 and AES-CCM-64-64-128 are mandatory to implement.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  data_1
]

data_1 = (
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  KID : bstr,
  ? APP_1 : bstr
)

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 4

- S_U - variable length session identifier

- N_U - 64-bit random nonce

- E_U - the ephemeral public key of Party U

- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference

- HKDFs_U - supported ECDH-SS w/ HKDF algorithms

- AEADs_U - supported AEAD algorithms

- KID - identifier of the pre-shared key

- APP_1 - bstr containing opaque application data

5.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_U.

- Choose a session identifier S_U which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party U. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_U SHALL be different from the concurrently used session identifiers of that protocol.

- Format message_1 as specified in Section 5.2.1.

5.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.

- Verify that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- For EC2 curves, validate that E_U is a valid point by verifying that there is a solution to the curve definition for the given parameter 'x'.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message SHOULD include a diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U.

- Pass APP_1 to the application.

5.3. EDHOC Message 2

5.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr
)

aad_2 : bstr
```

where aad_2, in non-CDDL notation, is:

```
aad_2 = H( message_1 | [ data_2 ] )
```

where:

- MSG_TYPE = 5
- S_V - variable length session identifier
- N_V - 64-bit random nonce
- E_V - the ephemeral public key of Party V
- HKDF_V - an single chosen algorithm from HKDFs_U
- AEAD_V - an single chosen algorithm from AEADs_U
- COSE_ENC_2 has the following fields and values:
  - external_aad = aad_2
  - plaintext = ? APP_2
- APP_2 - bstr containing opaque application data

- H() - the hash function in HKDF_V

5.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_V.

- Choose a session identifier S_V which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party V. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_V SHALL be different from the concurrently used session identifiers of that protocol.

- Select HKDF_V and AEAD_V from the algorithms proposed in HKDFs_U and AEADs_U.

- Format message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

5.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- For EC2 curves, validate that E_V is a valid point by verifying that there is a solution to the curve definition for the given parameter 'x'.

- Verify message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_2 to the application.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 : bstr
```

where aad_3, in non-CDDL notation, is:

```
aad_3 = H( H( message_1 | message_2 ) | [ data_3 ] )
```

where:

- MSG_TYPE = 6
- COSE_ENC_3 has the following fields and values:
  - external_aad = aad_3
  - plaintext = ? APP_3
- APP_3 - bstr containing opaque application data

5.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

5.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.

- o Verify message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is decrypted and verified as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- o Pass APP_3 to the application.

6. Error Handling

6.1. Error Message Format

This section defines a message format for an EDHOC error message, used during the protocol. This is an error on EDHOC level and is independent of the lower layers used. An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR array as defined below

```
error = [
  MSG_TYPE : int,
  ? ERR_MSG : tstr
]
```

where:

- o MSG_TYPE = 0

- o ERR_MSG is an optional text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252].

7. IANA Considerations

7.1. Media Types Registry

IANA has added the media type ’application/edhoc’ to the Media Types registry:

Type name: application

Subtype name: edhoc

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of this document.

Interoperability considerations: N/A

Published specification: [[this document]] (this document)

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

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

Person & email address to contact for further information:
Goeran Selander <goran.selander@ericsson.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander <goran.selander@ericsson.com>

Change Controller: IESG

8. Security Considerations

EDHOC builds on the SIGMA-I family of theoretical protocols that provides perfect forward secrecy and identity protection with a minimal number of messages. The encryption algorithm of the SIGMA-I protocol provides identity protection, but the security of the protocol requires the MAC to cover the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

EDHOC adds an explicit message type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. EDHOC uses the same Sign-then-MAC approach as TLS 1.3.

EDHOC does not include negotiation of parameters related to the ephemeral key, but it enables Party V to verify that the ECDH curve used in the protocol is the most preferred curve by U which is supported by both U and V.

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to APP_1 and APP_2 in the asymmetric case, and APP_1 and KID in the symmetric case. The communicating parties may therefore anonymize KID.

Using the same KID or unprotected application data in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported algorithms may be used to identify the application.

Party U and V are allowed to select the session identifiers S_U and S_V, respectively, for the other party to use in the ongoing EDHOC protocol as well as in a subsequent traffic protection protocol (e.g. OSCOAP). The choice of session identifier is not security critical but intended to simplify the retrieval of the right security context in combination with using short identifiers. If the wrong session identifier of the other party is used in a protocol message it will result in the receiving party not being able to retrieve a security context (which will terminate the protocol) or retrieving the wrong security context (which also terminates the protocol as the message cannot be verified).

Party U and V must make sure that unprotected data does not trigger any harmful actions. In particular, this applies to APP_1 in the asymmetric case, and APP_1 and KID in the symmetric case. Party V should be aware that replays of EDHOC message_1 cannot be detected unless previous nonces are stored.

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. If ECDSA is supported, "deterministic ECDSA" as specified in RFC6979 is RECOMMENDED.

Nonces MUST NOT be reused, both parties MUST generate fresh random nonces.

Ephemeral keys SHOULD NOT be reused, both parties SHOULD generate fresh random ephemeral key pairs. Party V MAY reuse the ephemeral key to limit the effect of certain DoS attacks. For example, to reduce processing costs in the case of repeated uncompleted protocol runs, party V MAY pre-compute its ephemeral key E_V and reuse it for a small number of concurrent EDHOC executions, for example until a number of EDHOC protocol instances has been successfully completed, which triggers party V to pre-compute a new ephemeral key E_V to use with subsequent protocol runs.

The referenced processing instructions in [SP-800-56a] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported.

The choice of key length used in the different algorithms needs to be harmonized, so that a sufficient security level is maintained for certificates, EDHOC, and the protection of application data. Party U and V should enforce a minimum security level.

Note that, depending on the application, the keys established through the EDHOC protocol will need to be renewed, in which case the communicating parties need to run the protocol again.

Implementations should provide countermeasures to side-channel attacks such as timing attacks.

9. Acknowledgments

The authors want to thank Dan Harkins, Ilari Liusvaara, Jim Schaad and Ludwig Seitz for reviewing intermediate versions of the draft and contributing concrete proposals incorporated in this version. We are especially indebted to Jim Schaad for his continuous reviewing and implementation of different versions of the draft.

TODO: This section should be after Appendices and before Authors' addresses according to RFC7322.

10. References

10.1. Normative References

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

[I-D.schaad-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Headers for carrying and referencing X.509 certificates", draft-schaad-cose-x509-01 (work in progress), May 2017.

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

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

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

[SIGMA] Krawczyk, H., "SIGMA - The ’SIGn-and-MAc’ Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols (Long version)", June 2003, <http://webee.technion.ac.il/˜hugo/sigma-pdf.pdf>.

[SP-800-56a]
Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2, May 2013, <http://dx.doi.org/10.6028/NIST.SP.800-56Ar2>.

10.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-02 (work in progress), January 2017.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-04 (work in progress), July 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[I-D.seitz-ace-oscoap-profile]
Seitz, L., Gunnarsson, M., and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-03 (work in progress), June 2017.

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

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

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

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. PSK Chaining

An application using EDHOC with symmetric keys may have a security policy to change the PSK as a result of successfully completing the EDHOC protocol. In this case, the old PSK SHALL be replaced with a new PSK derived using other = exchange_hash, AlgorithmID = "EDHOC PSK Chaining" and keyDataLength equal to the key length of AEAD_V, see Section 3.2.

Appendix C. EDHOC with CoAP and OSCOAP

C.1. Transferring EDHOC in CoAP

EDHOC can be transferred as an exchange of CoAP [RFC7252] messages, with the CoAP client as party U and the CoAP server as party V. By default EDHOC is sent to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discovered e.g. using resource directory [I-D.ietf-core-resource-directory].

In practice, EDHOC message_1 is sent in the payload of a POST request from the client to the server’s resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response. EDHOC message_3 or the EDHOC error message is sent from the client to the server’s resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response

An example of successful EDHOC exchange using CoAP is shown in Figure 5.

```
              Client    Server
                |          |
                +--------->| Header: POST (Code=0.02)
                |   POST   | Uri-Path: "/.well-known/edhoc"
                |          | Content-Type: application/edhoc
                |          | Payload: EDHOC message_1
                |          |
                |<---------+ Header: 2.04 Changed
                |   2.04   | Content-Type: application/edhoc
                |          | Payload: EDHOC message_2
                |          |
                +--------->| Header: POST (Code=0.02)
                |   POST   | Uri-Path: "/.well-known/edhoc"
                |          | Content-Type: application/edhoc
                |          | Payload: EDHOC message_3
                |          |
                |<---------+ Header: 2.04 Changed
                |   2.04   |
                |          |
```

Figure 5: Transferring EDHOC in CoAP

C.2. Deriving an OSCOAP context from EDHOC

When EDHOC is use to derive parameters for OSCOAP [I-D.ietf-core-object-security], the parties must make sure that the EDHOC session identifiers are unique Recipient IDs in OSCOAP. In case that the CoAP client is party U and the CoAP server is party V:

- The AEAD Algorithm is AEAD_V, as defined in this document

- The KDF algorithm is HKDF_V, as defined in this document

- The Client’s Sender ID is S_V, as defined in this document

- The Server’s Sender ID is S_U, as defined in this document

- The Master Secret is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Secret" and keyDataLength equal to the key length of AEAD_V.

- The Master Salt is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Salt" and keyDataLength equal to 64 bits.

Authors' Addresses

Goeran Selander
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

ACE Working Group   G. Selander
Internet-Draft   J. Mattsson
Intended status: Standards Track   F. Palombini
Expires: September 6, 2018   Ericsson AB
                                                          March 05, 2018

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-08

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs] or where the protocol needs to work on a variety of underlying protocols. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and energy [RFC7228]. A method for protecting individual messages at the application layer suitable for constrained devices, is provided by CBOR Object Signing and Encryption (COSE) [RFC8152]), which builds on the Concise Binary Object Representation (CBOR) [RFC7049].

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

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56a [SP-800-56a], and HKDF [RFC5869]. CBOR [RFC7049] and COSE [RFC8152] are used to implement these standards.

1.1. Terminology

This document use the same informational CBOR Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] grammar as COSE (see Section 1.3 of [RFC8152]). A vertical bar | denotes byte string concatenation.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

2. Protocol Overview

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and TLS 1.3, EDHOC is built on a variant of the SIGMA protocol which provide identity protection, and like TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an AEAD algorithm is shown in Figure 1.

```
      Party U                                                 Party V
         |                          E_U                          |
         +------------------------------------------------------>|
         |                                                       |
         |         E_V, Enc(K_2; ID_V, Sig(V; E_U, E_V);)        |
         |<------------------------------------------------------+
         |                                                       |
         |           Enc(K_3; ID_U, Sig(U; E_V, E_U);)           |
         +------------------------------------------------------>|
         |                                                       |
```

Figure 1: AEAD variant of the SIGMA-I protocol

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive the keying material. The messages are signed, MACed, and encrypted.

- E_U and E_V are the ECDH ephemeral public keys of U and V, respectively.

- ID_U and ID_V are identifiers for the public keys of U and V, respectively.

- Sig(U; .) and S(V; .) denote signatures made with the private key of U and V, respectively.

- Enc(K; P; A) denotes AEAD encryption of plaintext P and additional authenticated data A using the key K derived from the shared secret. The AEAD MUST NOT be replaced by plain encryption, see Section 8.

As described in Appendix B of [SIGMA], in order to create a "full-fledged" protocol some additional protocol elements are needed. EDHOC adds:

- Explicit session identifiers S_U, S_V different from other concurrent session identifiers (EDHOC or other used protocol identifier) chosen by U and V, respectively.

- Explicit nonces N_U, N_V chosen freshly and anew with each session by U and V, respectively.

- Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.

EDHOC also makes the following additions:

- Negotiation of key derivation, encryption, and signature algorithms:
    - U proposes one or more algorithms of the following kinds:
        - HKDF
        - AEAD
        - Signature verification
        - Signature generation
    - V selects one algorithm of each kind
- Verification of common preferred ECDH curve:
    - U lists supported ECDH curves in order of preference
    - V verifies that the ECDH curve of the ephemeral key is the most preferred common curve
- Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR and COSE libraries. EDHOC does not put any requirement on the lower layers and can therefore be also be used e.g. in environments without IP.

This paper is organized as follows: Section 3 specifies general properties of EDHOC, including formatting of the ephemeral public keys and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, and Appendix A provides a wealth of test vectors to ease implementation and ensure interoperability.

3. EDHOC Overview

EDHOC consists of three messages (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consists of a CBOR array where the first element is an int specifying the message type (MSG_TYPE). After creating EDHOC message_3, Party U can derive the traffic key (master secret) and protected application data can therefore be sent in parallel with EDHOC message_3. The application data may be protected using the negotiated AEAD algorithm and the explicit session identifiers S_U and S_V. EDHOC may be used with the media type application/edhoc defined in Section 7.

```
   Party U                                                 Party V
      |                                                       |
      | ------------------- EDHOC message_1 ----------------> |
      |                                                       |
      | <------------------ EDHOC message_2 ----------------- |
      |                                                       |
      | ------------------- EDHOC message_3 ----------------> |
      |                                                       |
      | <------------ Protected Application Data ------------> |
      |                                                       |
```

Figure 2: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or certificates (Cert). EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed.

EDHOC also allows opaque application data (APP_1, APP_2, APP_3) to be sent in the respective messages. APP_1 is unprotected, APP_2 is protected (encrypted and integrity protected), and APP_3 is protected and mutually authenticated. When EDHOC is used with asymmetric key authentication APP_2 is sent to an unauthenticated party, but with symmetric key authentication APP_2 is mutually authenticated.

3.1. Formatting of the Ephemeral Public Keys

The ECDH ephemeral public key SHALL be formatted as a COSE_Key of type EC2 or OKP according to section 13.1 and 13.2 of [RFC8152]. The curve X25519 is mandatory to implement. For Elliptic Curve Keys of type EC2, compact representation and compact output as per [RFC6090] SHALL be used, i.e. the ’y’ parameter SHALL NOT be present in the The COSE_Key object. COSE [RFC8152] always use compact output for Elliptic Curve Keys of type EC2.

3.2. Key Derivation

Key and IV derivation SHALL be done as specified in Section 11.1 of [RFC8152] with the following input:

- The PRF SHALL be the HKDF [RFC5869] in the ECDH-SS w/ HKDF negotiated during the message exchange (HKDF_V).

- The secret SHALL be the ECDH shared secret as defined in Section 12.4.1 of [RFC8152].

- The salt SHALL be the PSK when EDHOC is authenticated with symmetric keys and the empty string "" when EDHOC is authenticated with asymmetric keys.

- The fields in the context information COSE_KDF_Context SHALL have the following values:

  - AlgorithmID is an int or tstr as defined below

  - PartyUInfo = PartyVInfo = (nil, nil, nil)

  - keyDataLength is a uint as defined below

  - protected SHALL be a zero length bstr

  - other is a bstr SHALL be aad_2, aad_3, or exchange_hash

where exchange_hash, in non-CDDL notation, is:

```
+---------------------------------+-------------+-------------+
| exchange_hash = H( H( message_1 | message_2 ) | message_3 ) |
+---------------------------------+-------------+-------------+
+---------------------------------+-------------+-------------+
```

where H() is the hash function in HKDF_V.

For message_i the key, called K_i, SHALL be derived using other = aad_i, where i = 2 or 3. The key SHALL be derived using AlgorithmID set to the integer value of the negotiated AEAD (AEAD_V), and keyDataLength equal to the key length of AEAD_V.

If the AEAD algorithm requires an IV, then IV_i for message_i SHALL be derived using other = aad_i, where i = 2 or 3. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in section 12.1.2. of [RFC8152], and keyDataLength equal to the IV length of AEAD_V.

Application specific traffic keys and other data SHALL be derived using other = exchange_hash. AlgorithmID SHALL be a tstr defined by the application and SHALL be different for different data being derived (an example is given in Appendix C.2). keyDataLength is set to the length of the data being derived.

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

EDHOC supports authentication with raw public keys (RPK) and certificates (Cert) with the requirements that:

- Party U SHALL be able to identify Party V’s public key using ID_V.

- Party V SHALL be able to identify Party U’s public key using ID_U.

ID_U and ID_V SHALL either contain the credential used for authentication (e.g. x5bag or x5chain) or uniquely identify the credential used for authentication (e.g. x5t), see [I-D.schaad-cose-x509]. Party U and V MAY retrieve the other party’s credential out of band. Optionally, ID_U and ID_V are complemented with the additional parameters HINT_ID_U and HINT_ID_V containing information about how to retrieve the credential of Party U and Party V, respectively (e.g. x5u), see [I-D.schaad-cose-x509].

Party U and Party V MAY use different type of credentials, e.g. one uses RPK and the other uses Cert. Party U and Party V MAY use different signature algorithms.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

```
Party U                                                          Party V
|                      S_U, N_U, E_U, ALG_1, APP_1                     |
+--------------------------------------------------------------------->|
|                               message_1                              |
|                                                                      |
|S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; Sig(V; ID_V, aad_2, APP_2); aad_2)|
|<---------------------------------------------------------------------+
|                               message_2                              |
|                                                                      |
|           S_V, Enc(K_3; Sig(U; ID_U, aad_3, APP_3); aad_3)           |
+--------------------------------------------------------------------->|
|                               message_3                              |
```

Figure 3: EDHOC with asymmetric key authentication.

4.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with asymmetric keys, the COSE algorithms ECDH-SS + HKDF-256, AES-CCM-64-64-128, and EdDSA are mandatory to implement.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  SIGs_V : alg_array,
  SIGs_U : alg_array,
  ? APP_1 : bstr
]

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 1
- S_U - variable length session identifier
- N_U - 64-bit random nonce
- E_U - the ephemeral public key of Party U
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- HKDFs_U - supported ECDH-SS w/ HKDF algorithms
- AEADs_U - supported AEAD algorithms
- SIGs_V - signature algorithms, with which Party U supports verification
- SIGs_U - signature algorithms, with which Party U supports signing
- APP_1 - bstr containing opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used. The content of ECDH-Curves_U SHALL be fixed, and SHALL NOT be changed based on previous error messages.

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_U.

- Choose a session identifier S_U which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other concurrent session identifiers used by Party U. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_U SHALL be different from the concurrently used session identifiers of that protocol.

- Format message_1 as specified in Section 4.2.1.

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.

- Verify that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- For elliptic curves, that E_U is a valid point by verifying that there is a solution to the curve definition for the given parameter ’x’.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message MUST include the following diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U:

```
ERR_MSG = "Curve not supported; X"
```

where X is the first curve in ECDH-Curves_U that V supports, encoded as in Table 22 of {{RFC8152}}.

o Pass APP_1 to the application.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr,
  SIG_V : int / tstr,
  SIG_U : int / tstr
)

aad_2 : bstr
```

where aad_2, in non-CDDL notation, is:

```
aad_2 = H( message_1 | [ data_2 ] )
```

where:

o MSG_TYPE = 2

o S_V - variable length session identifier

o N_V - 64-bit random nonce

o E_V - the ephemeral public key of Party V

- HKDF_V - a single chosen algorithm from HKDFs_U

- AEAD_V - a single chosen algorithm from AEADs_U

- SIG_V - a single chosen algorithm from SIGs_V with which Party V signs

- SIG_U - a single chosen algorithm from SIGs_U with which Party U signs

- COSE_ENC_2 has the following fields and values:

  * external_aad = aad_2

  * plaintext = [COSE_SIG_V, ? APP_2]

- COSE_SIG_V is a COSE_Sign1 object with the following fields and values:

  * protected = { abc : ID_V, ? xyz : HINT_ID_V }

  * detached payload = aad_2, ? APP_2

- abc - any COSE map label that can identify a public key, see Section 4.1

- ID_V - identifier for the public key of Party V

- xyz - any COSE map label for information about how to retrieve the credential of Party V, see Section 4.1

- HINT_ID_V - information about how to retrieve the credential of Party V

- APP_2 - bstr containing opaque application data

- H() - the hash function in HKDF_V

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_V.

- Choose a session identifier S_V which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party V. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_V SHALL be different from the concurrently used session identifiers of that protocol.

- Select HKDF_V, AEAD_V, SIG_V, and SIG_U from the algorithms proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- Format message_2 as specified in Section 4.3.1:

  - COSE_Sign1 is computed as defined in section 4.4 of [RFC8152], using algorithm SIG_V and the private key of Party V.

  - COSE_Encrypt0 is computed as defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- Verify that HKDF_V, AEAD_V, SIG_V, and SIG_U were proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- Verify (OPTIONAL) that N_V has not been received before.

- For elliptic curves, validate that E_V is a valid point by verifying that there is a solution to the curve definition for the given parameter ’x’.

- Verify message_2 as specified in Section 4.3.1:

  - COSE_Encrypt0 is decrypted defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2.

  - COSE_Sign1 is verified as defined in section 4.4 of [RFC8152], using algorithm SIG_V and the public key of Party V.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_2 to the application.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 : bstr
```

where aad_3, in non-CDDL notation, is:

```
aad_3 = H( H( message_1 | message_2 ) | [ data_3 ] )
```

where:

- MSG_TYPE = 3
- COSE_ENC_3 has the following fields and values:
  - external_aad = aad_3
  - plaintext = [COSE_SIG_U, ? APP_3]
- COSE_SIG_U is a COSE_Sign1 object with the following fields and values:
  - protected = { abc : ID_U, ? xyz : HINT_ID_U }
  - detached payload = aad_3, ? APP_3
- abc - any COSE map label that can identify a public key, see Section 4.1
- ID_U - identifier for the public key of Party U
- xyz - any COSE map label for information about how to retrieve the credential of Party U, see Section 4.1

- HINT_ID_V - information about how to retrieve the credential of Party U

- APP_3 - bstr containing opaque application data

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 4.4.1:

  * COSE_Sign1 is computed as defined in section 4.4 of [RFC8152], using algorithm SIG_U and the private key of Party U.

  * COSE_Encrypt0 is computed as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.

- Verify message_3 as specified in Section 4.4.1:

  * COSE_Encrypt0 is decrypted as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3.

  * COSE_Sign1 is verified as defined in section 4.4 of [RFC8152], using algorithm SIG_U and the public key of Party U.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_3 to the application.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

EDHOC supports authentication with pre-shared keys. Party U and V are assumed to have a pre-shared uniformly random key (PSK) with the requirement that:

- Party V SHALL be able to identify the PSK using KID.

KID may optionally contain information about how to retrieve the PSK.

EDHOC with symmetric key authentication is illustrated in Figure 4.

```
   Party U                                                       Party V
   |                S_U, N_U, E_U, ALG_1, KID, APP_1                   |
   +------------------------------------------------------------------>|
   |                             message_1                             |
   |                                                                   |
   |         S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; APP_2; aad_2)         |
   |<------------------------------------------------------------------+
   |                             message_2                             |
   |                                                                   |
   |                    S_V, Enc(K_3; APP_3; aad_3)                    |
   +------------------------------------------------------------------>|
   |                             message_3                             |
```

Figure 4: EDHOC with symmetric key authentication.

5.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with symmetric keys, the COSE algorithms ECDH-SS + HKDF-256 and AES-CCM-64-64-128 are mandatory to implement.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  data_1
]

data_1 = (
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  KID : bstr,
  ? APP_1 : bstr
)

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 4

- S_U - variable length session identifier

- N_U - 64-bit random nonce

- E_U - the ephemeral public key of Party U

- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference

- HKDFs_U - supported ECDH-SS w/ HKDF algorithms

- AEADs_U - supported AEAD algorithms

- KID - identifier of the pre-shared key

- APP_1 - bstr containing opaque application data

5.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_U.

- Choose a session identifier S_U which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party U. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_U SHALL be different from the concurrently used session identifiers of that protocol.

- Format message_1 as specified in Section 5.2.1.

5.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Verify (OPTIONAL) that N_U has not been received before.

- Verify that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- For elliptic curves, validate that E_U is a valid point by verifying that there is a solution to the curve definition for the given parameter 'x'.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message SHOULD include a diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U.

- Pass APP_1 to the application.

5.3. EDHOC Message 2

5.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr
)

aad_2 : bstr
```

where aad_2, in non-CDDL notation, is:

```
aad_2 = H( message_1 | [ data_2 ] )
```

where:

- MSG_TYPE = 5
- S_V - variable length session identifier
- N_V - 64-bit random nonce
- E_V - the ephemeral public key of Party V
- HKDF_V - an single chosen algorithm from HKDFs_U
- AEAD_V - an single chosen algorithm from AEADs_U
- COSE_ENC_2 has the following fields and values:
  - external_aad = aad_2
  - plaintext = ? APP_2
- APP_2 - bstr containing opaque application data

- H() - the hash function in HKDF_V

5.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_V.

- Choose a session identifier S_V which is not in use and store it for the length of the protocol. The session identifier SHOULD be different from other relevant concurrent session identifiers used by Party V. The session identifier MAY be used with the protocol for which EDHOC establishes traffic keys/master secret, in which case S_V SHALL be different from the concurrently used session identifiers of that protocol.

- Select HKDF_V and AEAD_V from the algorithms proposed in HKDFs_U and AEADs_U.

- Format message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2.

5.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- For elliptic curves, validate that E_V is a valid point by verifying that there is a solution to the curve definition for the given parameter 'x'.

- Verify message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is decrypted defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_2 to the application.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 : bstr
```

where aad_3, in non-CDDL notation, is:

```
aad_3 = H( H( message_1 | message_2 ) | [ data_3 ] )
```

where:

- MSG_TYPE = 6

- COSE_ENC_3 has the following fields and values:

  * external_aad = aad_3

  * plaintext = ? APP_3

- APP_3 - bstr containing opaque application data

5.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3.

5.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.

- Verify message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is decrypted and verified as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

- Pass APP_3 to the application.

6. Error Handling

6.1. Error Message Format

This section defines a message format for an EDHOC error message, used during the protocol. This is an error on EDHOC level and is independent of the lower layers used. An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR array as defined below

```
error = [
  MSG_TYPE : int,
  ? ERR_MSG : tstr
]
```

where:

- MSG_TYPE = 0

- ERR_MSG is an optional text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252].

7. IANA Considerations

7.1. The Well-Known URI Registry

IANA has added the well-known URI 'edhoc' in the Well-Known URIs registry.

URI suffix: edhoc

Change controller: IETF

Specification document(s): [[this document]]

Related information: None

7.2. Media Types Registry

IANA has added the media type 'application/edhoc' to the Media Types registry:

Type name: application

Subtype name: edhoc

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of this document.

Interoperability considerations: N/A

Published specification: [[this document]] (this document)

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

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

Person & email address to contact for further information:
Goeran Selander <goran.selander@ericsson.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander <goran.selander@ericsson.com>

Change Controller: IESG

8. Security Considerations

EDHOC builds on the SIGMA-I family of theoretical protocols that provides perfect forward secrecy and identity protection with a minimal number of messages. The encryption algorithm of the SIGMA-I protocol provides identity protection, but the security of the protocol requires the MAC to cover the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

EDHOC adds an explicit message type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. EDHOC uses the same Sign-then-MAC approach as TLS 1.3.

EDHOC does not include negotiation of parameters related to the ephemeral key, but it enables Party V to verify that the ECDH curve used in the protocol is the most preferred curve by U which is supported by both U and V.

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to APP_1 and APP_2 in the asymmetric case, and APP_1 and KID in the symmetric case. The communicating parties may therefore anonymize KID.

Using the same KID or unprotected application data in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported algorithms may be used to identify the application.

Party U and V are allowed to select the session identifiers S_U and S_V, respectively, for the other party to use in the ongoing EDHOC protocol as well as in a subsequent traffic protection protocol (e.g. OSCORE [I-D.ietf-core-object-security]). The choice of session identifier is not security critical but intended to simplify the retrieval of the right security context in combination with using short identifiers. If the wrong session identifier of the other party is used in a protocol message it will result in the receiving party not being able to retrieve a security context (which will terminate the protocol) or retrieving the wrong security context (which also terminates the protocol as the message cannot be verified).

Party U and V must make sure that unprotected data does not trigger any harmful actions. In particular, this applies to APP_1 in the asymmetric case, and APP_1 and KID in the symmetric case. Party V should be aware that replays of EDHOC message_1 cannot be detected unless previous nonces are stored.

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. If ECDSA is supported, "deterministic ECDSA" as specified in RFC6979 is RECOMMENDED.

Nonces MUST NOT be reused, both parties MUST generate fresh random nonces.

Ephemeral keys SHOULD NOT be reused, both parties SHOULD generate fresh random ephemeral key pairs. Party V MAY reuse the ephemeral key to limit the effect of certain DoS attacks. For example, to reduce processing costs in the case of repeated uncompleted protocol runs, party V MAY pre-compute its ephemeral key E_V and reuse it for a small number of concurrent EDHOC executions, for example until a number of EDHOC protocol instances has been successfully completed, which triggers party V to pre-compute a new ephemeral key E_V to use with subsequent protocol runs.

The referenced processing instructions in [SP-800-56a] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported.

The choice of key length used in the different algorithms needs to be harmonized, so that a sufficient security level is maintained for certificates, EDHOC, and the protection of application data. Party U and V should enforce a minimum security level.

Note that, depending on the application, the keys established through the EDHOC protocol will need to be renewed, in which case the communicating parties need to run the protocol again.

Implementations should provide countermeasures to side-channel attacks such as timing attacks.

9. Acknowledgments

The authors want to thank Dan Harkins, Ilari Liusvaara, Jim Schaad and Ludwig Seitz for reviewing intermediate versions of the draft and contributing concrete proposals incorporated in this version. We are especially indebted to Jim Schaad for his continuous reviewing and implementation of different versions of the draft.

TODO: This section should be after Appendices and before Authors' addresses according to RFC7322.

10. References

10.1. Normative References

[I-D.schaad-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Headers for carrying and referencing X.509 certificates", draft-schaad-cose-x509-01 (work in progress), May 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>.

[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>.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[SIGMA] Krawczyk, H., "SIGMA - The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols (Long version)", June 2003, <http://webee.technion.ac.il/˜hugo/sigma-pdf.pdf>.

[SP-800-56a]
Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2, May 2013, <http://dx.doi.org/10.6028/NIST.SP.800-56Ar2>.

10.2. Informative References

[I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 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-10 (work in progress), February 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-00 (work in progress), December 2017.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 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-08 (work in progress), January 2018.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-12 (work in progress), October 2017.

[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>.

[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>.

[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>.

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. PSK Chaining

An application using EDHOC with symmetric keys may have a security policy to change the PSK as a result of successfully completing the EDHOC protocol. In this case, the old PSK SHALL be replaced with a new PSK derived using other = exchange_hash, AlgorithmID = "EDHOC PSK Chaining" and keyDataLength equal to the key length of AEAD_V, see Section 3.2.

Appendix C. EDHOC with CoAP and OSCORE

C.1. Transferring EDHOC in CoAP

EDHOC can be transferred as an exchange of CoAP [RFC7252] messages, with the CoAP client as party U and the CoAP server as party V. By default EDHOC is sent to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discovered e.g. using resource directory [I-D.ietf-core-resource-directory].

In practice, EDHOC message_1 is sent in the payload of a POST request from the client to the server’s resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response. EDHOC message_3 or the EDHOC error message is sent from the client to the server’s resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response

An example of successful EDHOC exchange using CoAP is shown in Figure 5.

```
              Client    Server
                |          |
                +--------->| Header: POST (Code=0.02)
                |   POST   | Uri-Path: "/.well-known/edhoc"
                |          | Content-Type: application/edhoc
                |          | Payload: EDHOC message_1
                |          |
                |<---------+ Header: 2.04 Changed
                |   2.04   | Content-Type: application/edhoc
                |          | Payload: EDHOC message_2
                |          |
                +--------->| Header: POST (Code=0.02)
                |   POST   | Uri-Path: "/.well-known/edhoc"
                |          | Content-Type: application/edhoc
                |          | Payload: EDHOC message_3
                |          |
                |<---------+ Header: 2.04 Changed
                |   2.04   |
                |          |
```

Figure 5: Transferring EDHOC in CoAP

C.2. Deriving an OSCORE context from EDHOC

When EDHOC is use to derive parameters for OSCORE [I-D.ietf-core-object-security], the parties must make sure that the EDHOC session identifiers are unique Recipient IDs in OSCORE. In case that the CoAP client is party U and the CoAP server is party V:

- The AEAD Algorithm is AEAD_V, as defined in this document

- The Key Derivation Function (KDF) is HKDF_V, as defined in this document

- The Client’s Sender ID is S_V, as defined in this document

- The Server’s Sender ID is S_U, as defined in this document

- The Master Secret is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCORE Master Secret" and keyDataLength equal to the key length of AEAD_V.

- The Master Salt is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCORE Master Salt" and keyDataLength equal to 64 bits.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 12, 2018

C. Sengul
A. Kirby
Nominet
P. Fremantle
University of Portsmouth

April 10, 2018

MQTT-TLS profile of ACE
draft-sengul-ace-mqtt-tls-profile-02

Abstract

This document specifies a profile for the ACE (Authentication and Authorization for Constrained Environments) to enable authorization in an MQTT-based publish-subscribe messaging system. Proof-of-possession keys, bound to OAuth2.0 access tokens, are used to authenticate and authorize publisher and subscriber clients. The protocol relies on TLS for confidentiality and server authentication.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies a profile for the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, clients and a resource server use MQTT to communicate. The protocol relies on TLS for communication security between entities. The basic protocol interactions follow MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard]. In addition, this document describes improvements to the basic protocol with the new MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5] (e.g., improved authentication exchange and error reporting). Both versions are expected to be supported in practice, and therefore, covered in this document.

MQTT is a publish-subscribe protocol and supports two types of client operation: publish and subscribe. Once connected, a client can publish to multiple topics, and subscribe to multiple topics; however, for the purpose of this document, these actions are described separately. The MQTT broker is responsible for distributing messages published by the publishers to the appropriate subscribers. Each publish message contains a topic, which is used by the broker to filter the subscribers for the message. Subscribers must subscribe to the topics to receive the corresponding messages.

In this document, message topics are treated as resources. Clients use an access token, bound to a key (the proof-of-possession key) to authorize with the MQTT broker their connection and publish/subscribe permissions to topics. In the context of this ACE profile, the MQTT broker acts as the resource server. In order to provide communication confidentiality and resource server authentication, TLS is used.

Clients use client authorization servers [I-D.ietf-ace-actors] to obtain tokens from the authorization server. The communication protocol between the client authorization server and the authorization server is assumed to be HTTPS. Also, if the broker supports token introspection, it is assumed to use HTTPS to communicate with the authorization server. These interfaces MAY be implemented using other protocols e.g., CoAP or MQTT. This document makes the same assumptions as the Section 4 of the ACE framework [I-D.ietf-ace-oauth-authz] in terms of client and RS registration with the AS and establishing of keying material.

This document describes authorization of the following exchanges between publisher and subscriber clients, and the broker.

- Connection establishment between the clients and the broker

- Publish messages from the publishers to the broker, and from the broker to the subscribers

- Subscribe messages from the subscribers to the broker

In Section 2, these exchanges are described based on the MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard]. These exchanges are also supported by the new MQTT v5 - the OASIS Specification Draft

[MQTT-OASIS-Standard-v5]. Section 3 describes how they may be improved by the new MQTT v5.

1.1. Requirements Language

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

1.2. ACE-Related Terminology

The terminology for entities in the architecture is defined in OAuth 2.0 RFC 6749 [RFC6749] and ACE actors [I-D.ietf-ace-actors], such as "Client" (C), "Resource Server" (RS) and "Authorization Server" (AS).

The term "endpoint" is used following its OAuth definition, to denote resources such as /token and /introspect at the AS.

The term "Resource" is used to refer to an MQTT "topic", which is defined in Section 1.2. Hence, the "Resource Owner" is any entity that can authoritatively speak for the "topic".

Certain security-related terms such as "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify" are taken from RFC 4949 [RFC4949].

1.3. MQTT-Related Terminology

The document describes message exchanges as MQTT protocol interactions. For additional information, please refer to the MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard] or the MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5].

Topic name
: The label attached to an application message, which is matched to a subscription.

Topic filter
: An expression that indicates interest in one or more topic names. Topic filters may include wildcards.

Subscription
: A subscription comprises a Topic filter and a maximum quality of service (QoS).

Application Message
: The data carried by the MQTT protocol. The data has an associated QoS level and a Topic name.

MQTT sends various control messages across a network connection. The following is not an exhaustive list and the control packets that are not relevant for authorization are not explained. These include, for instance, the PUBREL and PUBCOMP packets used in the 4-step handshake required for the QoS level 2.

CONNECT
: Client request to connect to the broker. After a network connection is established, this is the first packet sent by a client.

CONNACK
: The broker connection acknowledgment. The first packet sent from the broker to a client is a CONNACK packet. CONNACK packets contain return codes indicating either a success or an error state to a client.

PUBLISH
: Publish packet that can be sent from a client to the broker, or from the broker to a client.

PUBACK
: Response to PUBLISH packet with QoS level 1. PUBACK can be sent from the broker to a client or a client to the broker.

PUBREC
: Response to PUBLISH packet with QoS level 2. PUBREC can be sent from the broker to a client or a client to the broker.

SUBSCRIBE
: The client subscribe request.

SUBACK
: Subscribe acknowledgment.

PINGREQ
: A ping request sent from a client to the broker. It signals to the broker that the client is alive, and is used to confirm that the broker is still alive.

2. Basic Protocol Interactions

This section describes the following exchanges between publisher and subscriber clients, the broker, and the authorization server according to the MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard]. These exchanges are compatible also with the new MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5]. In addition, Section 3 describes how these exchanges may be improved with the MQTT v5.

- Authorizing connection establishment between the clients and the broker

- Authorizing publish messages from the publishers to the broker, and from the broker to the subscribers

- Authorizing subscribe messages from the subscribers to the broker

Message topics are treated as resources. The publisher and subscriber clients are assumed to have identified the topics of interest out-of-band (topic discovery is not a feature of the MQTT protocol).

A connection request carries a token specifying the permissions that the client has (e.g., publish permission to a given topic). A resource owner can pre-configure policies at the AS that give clients publish or subscribe permissions to different topics.

2.1. Authorizing Connection Establishment

This section specifies how publishers and subscribers establish an authorized connection to an MQTT broker. The token request and response use the /token endpoint of the authorization server, as specified in Section 6 of the ACE framework [I-D.ietf-ace-oauth-authz].

Figure 1 shows the basic protocol flow during connection establishment.

```
                               +----------------+
      +---(A) Token request----| Client         |
      |                        | Authorization  |
      |   +-(B) Access token-->| Server         |
      |   |                    |________________|
      |   |                            |
      |   |              (C) Client On-boarding
      |   |                            |
      |   |                  +---------v-----+
   +--v-------------+        | Publisher or  |
   |                |        | Subscriber    |
   |  Authorization |        |_______________|
   |  Server        |            |       ^
   |________________|            |       |
      |    ^             (D)Connection  (G)Connection
      |    |               request +    response
      |    |               access token  |
      |    |                     |       |
      |    |                 +---v--------------+
      |    |                 |   Broker         |
      |    +(E)Introspection-| Resource Server  |
      |     request          |                  |
      +-(F)Introspection---->|__________________|
           response
```

Figure 1: Connection establishment

2.1.1. Client Authorization Server (CAS) and Authorization Server (AS) Interaction

The first step in the protocol flow (Figure 1 (A)) is token acquisition by the client authorization server (CAS) from the AS. If a client has enough resources and can support HTTPS, or optionally the AS supports MQTTS, these steps can instead be carried out by a client directly.

When requesting an access token from the AS, the CAS MAY include parameters in its request as defined in Section 6.1 of the ACE framework [I-D.ietf-ace-oauth-authz]. The content type is set to "application/json". The profile name is 'mqtt_tls'.

If the access token request has been successfully verified by the AS and the client is authorized to obtain a token for the indicated audience (e.g., topics) and scopes (e.g., publish/subscribe permissions), the AS issues an access token (Figure 1 (B)). The response includes the parameters described in Section 6.2 of the ACE framework [I-D.ietf-ace-oauth-authz]. This includes a token, which is assumed to be PoP by default. Hence, a 'cnf' parameter with a symmetric or asymmetric PoP key is returned. The token may be a reference, or a CBOR or JWT web token. Note that the 'cnf' parameter in the web tokens are to be consumed by the resource server and not the client. For more information on Proof of Possession semantics in JWTs see RFC 7800 [RFC7800] and for CWTs, see Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs) [I-D.ietf-ace-cwt-proof-of-possession].

In the case of an error, the AS returns error responses for HTTP-based interactions as ASCII codes in JSON content, as defined in Section 5.2 of RFC 6749 [RFC6749].

2.1.2. Client connection request to the broker

Client on-boarding (Figure 1 (C)) is out of the scope of this document. Once the client acquires the token, it can use it to request an MQTT connection to the broker over a TLS session with server authentication (Figure 1 (D)). This section describes the client transporting the token to the broker (RS) via the CONNECT control message after the TLS handshake. This is similar to an earlier proposal by Fremantle et al. [fremantle14]. An improvement to this is presented in Section 3 for the MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5]. Alternatively, the token may be used for the TLS session establishment as described in the DTLS profile for ACE [I-D.gerdes-ace-dtls-authorize]. In this case, both the TLS PSK and RPK handshakes MAY be supported. This may additionally require that the client transports the token to the broker before the connection establishment. To this end, the broker MAY support /authz-info endpoint via the "authz-info" topic. Then, to transport the token, clients publish to "authz-info" topic unauthorized. The topic "authz-info" MUST be publish-only for clients (i.e., the clients are not allowed to subscribe to it). This option is described in more detail in Appendix B.

When the client wishes to connect to the broker, it uses the CONNECT message of MQTT. Figure 2 shows the structure of the MQTT CONNECT control message.

```
 0            8             16            24            32
 +--------------------------------------------------------+
 |CPT=1 | Rsvd.|Remaining len.| Protocol  name len. = 4 |
 +--------------------------------------------------------+
 |                      'M' 'Q' 'T' 'T'                  |
 +--------------------------------------------------------+
 | Proto.level=4|Connect flags|          Keep alive     |
 +--------------------------------------------------------+
 |         Payload including User Name (='token')        |
 |     Password length and data (=signature/MAC)         |
 |                           ...                         |
 +--------------------------------------------------------+
```

Figure 2: MQTT CONNECT control message. (CPT=Control Packet Type, Rsvd=Reserved, len.=length, Proto.=Protocol)

To communicate the necessary connection parameters, the Client uses the appropriate flags of the CONNECT message. Figure 3 shows how the MQTT connect flags MUST be set to initiate a connection with the broker.

User name flag	Pass. flag	Will retain	Will QoS	Will Flag	Clean	Rsvd.
1	1	X	X X	X	1	0

Figure 3: MQTT CONNECT flags. (Rsvd=Reserved)

In order to ensure that the client and the broker discard any previous session and start a new session, the Clean Session Flag MUST be set to 1.

The Will flag indicates that a Will message needs to be sent when a client disconnection occurs. The situations in which the Will message is published include disconnections due to I/O or network failures, and the server closing the networking connection due to a protocol error. The client may set the Will flag as desired (marked as 'X' in Figure 3). If the Will flag is set to 1 and the broker accepts the connection request, the broker must store the Will message, and publish it when the network connection is closed according to Will QoS and Will retain parameters, and MQTT Will management rules. Section 2.5 explains how the broker deals with the retained messages in further detail.

Finally, Username and Password flags MUST be set to 1 to ensure that the Payload of the CONNECT message includes both Username and Password fields.

The CONNECT message defaults to ACE for authentication and authorization. For the basic operation described in this section, the Username field MUST be set to the token. The Password field MUST be set to the keyed message digest (MAC) or signature. The client MAY apply the PoP key either to the token or the entire request by computing a keyed message digest (for symmetric key) or a digital signature (for asymmetric key). (The Username field is a UTF-8 encoded string, which is prefixed with a two-byte length field and can have any length in the range of 0 and 65535. Similarly, the password field contains 0 to 65535 bytes of binary data, prefixed by a two-byte length field.)

2.1.3. Token validation

RS MUST verify the validity of the token. This validation MAY be done locally (e.g., in the case of a self-contained token) or the RS MAY send an introspection request to the AS. If introspection is used, this section follows similar steps to those described in Sections 7.2 and 7.3 of the ACE framework [I-D.ietf-ace-oauth-authz]. The communication between AS and RS MAY be HTTPS, but it, in every case, MUST be confidential, mutually authenticated and integrity protected.

The broker MUST check if the token is active either using ’expires_in’ parameter of the token or ’active’ parameter of the introspection response.

The access token is constructed by the AS such that RS can associate the access token with the client key. This document assumes that the Access Token is a PoP token as described in [I-D.ietf-ace-oauth-authz]. Therefore, the necessary information is contained in the ’cnf’ claim of the access token and may use either public or shared key approaches. The client uses the signature or the MAC in the password field to prove the possession of the key. Depending on the chosen implementation, the resource server validates the signature or the MAC over the token or the contents of the packet, authenticating the client.

The broker uses the scope field in the token (or in the introspection result) to determine the publish and subscribe permissions for the client. If the Will flag is set, then the broker MUST check that the token allows the publication of the Will message too.

The broker MAY cache the introspection result because it will need to decide whether to accept subsequent PUBLISH and SUBSCRIBE messages and these messages, which are sent after a connection is set-up, do not contain tokens. If the introspection result is not cached, then the RS needs to introspect the saved token for each request.

Note: Scope strings MAY follow an application specific convention. One option is to encode the permission and the topics it applies to the scope string e.g., ’publish_topic1’ or ’subscribe_topic2’. A second option is to simply use the keywords ’publish’ or ’subscribe’ as scope strings and use the ’aud’ field to define the topic. Another option is to use topic names as scope strings and use the ’aud’ field to define whether the ’publish’ or ’subscribe’ permission applies to these scopes. The choice is left to the implementer and depends on how the following trade-off is expected to be handled: token simplicity versus the number of tokens the broker is expected to handle per client.

2.1.4. The broker’s response to client connection request

Based on the validation result (obtained either via local inspection or using the /introspection interface of the AS), the broker MUST send a CONNACK message to the client.

The broker responses may follow either the MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard] or the MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], depending on which version(s) the broker supports.

In MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard], it is not possible to support AS discovery via sending a tokenless CONNECT message to the broker. This is because a CONNACK packet does not include a means to provide additional information to the client. Therefore, AS discovery needs to take place out-of-band. This is remedied in the MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5] and a solution is described in Section 3.

If the RS accepts the connection, it MUST store the token.

2.2. Authorizing PUBLISH messages

2.2.1. PUBLISH messages from the publisher client to the broker

On receiving the PUBLISH message, the broker MUST use the type of message (i.e., PUBLISH) and the topic name in the message header to compare against the cached token or its introspection result (depending on the implementation, different fields of the token or the introspection result may be checked, see the Note in Section 2.1.3).

If the client is allowed to publish to the topic, the RS must publish the message to all valid subscribers of the topic. The broker may also return an acknowledgment message if the QoS level is greater than or equal to 1.

In case of a failure, it is not possible to return an error in MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard]. The return of acknowledgement messages only indicates success. In the case of an authorization error, the broker SHOULD disconnect the client. Otherwise, it MUST ignore the PUBLISH message. Also, DISCONNECT messages are only sent from a client to the broker. So, server disconnection needs to take place below the application layer. However, in MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], it is possible to indicate failure and provide a reason code. Section 3 describes in more detail how PUBLISH authorization errors are handled.

2.2.2. PUBLISH messages from the broker to the subscriber clients

To forward PUBLISH messages to the subscribing clients, the broker identifies all the subscribers that have matching valid topic subscriptions (i.e., the tokens are valid and token scopes allow a subscription to the particular topic name). The broker sends a PUBLISH message with the topic name and the topic message to all the valid subscribers.

In MQTT, after connection establishment, there is no way to inform a client that an authorization error has occurred for previously subscribed topics, e.g., token expiry. In the case of an authorization error, the broker has two options: (1) stop forwarding PUBLISH messages to the unauthorized client or (2) disconnect the client. In the MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard], the MQTT DISCONNECT messages are only sent from a client to the broker. Therefore, the server disconnection needs to take place below the application layer. In MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], a server-side DISCONNECT message is possible and described in Section 3.

2.3. Authorizing SUBSCRIBE messages

In MQTT, a SUBSCRIBE message is sent from a client to the broker to create one or more subscriptions to one or more topics. The SUBSCRIBE message may contain multiple topic filters. The topic filters may include wildcard characters.

On receiving the SUBSCRIBE message, the broker MUST use the type of message (i.e., SUBSCRIBE) and the topic filter in the message header to compare against the stored token or introspection result (depending on the implementation, different fields of the token or introspection result may be checked, see the Note in Section 2.1.3).

As a response to the SUBSCRIBE message, the broker issues a SUBACK message. For each topic filter, the SUBACK packet includes a return code matching the QoS level for the corresponding topic filter. In the case of failure, the return code, in MQTT v3.1, must be 0x80 indicating 'Failure'. In MQTT v5, the appropriate return code is 0x87, indicating that the client is 'Not authorized'. Note that, in both MQTT versions, a reason code is returned for each topic filter. Therefore, the client may receive success codes for a subset of its topic filters, while being unauthorized for the rest.

2.4. Token expiration

The broker MUST check for token expiration whenever a CONNECT, PUBLISH or SUBSCRIBE message is received or sent. The broker SHOULD check for token expiration on receiving a PINGREQUEST message. This may allow for early detection of a token expiry.

The token validation is done either by checking the 'exp' claim of a CWT/JWT or via performing an introspection request with the Authorization server as described in Section 8.2 of the ACE framework [I-D.ietf-ace-oauth-authz]. In the basic operation, token expirations MAY lead to disconnecting the associated client. However, in MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], better error handling and re-authentication are possible. This is explained in more detail in Section 3.

2.5. Handling disconnections and retained messages

According to MQTT v3.1 - the OASIS Standard [MQTT-OASIS-Standard], only Client DISCONNECT messages are allowed. In MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], server-side DISCONNECT messages are possible, allowing to return '0x87 Not Authorized' return code to the client.

In the case of a DISCONNECT, due to the Clean Session flag, the broker deletes all session state but MUST keep the retained messages. By setting a RETAIN flag in a PUBLISH message the publisher indicates to the broker that it should store the most recent message for the associated topic. Hence, the new subscribers can receive the last sent message from the publisher for that particular topic without waiting for the next PUBLISH message. In the case of a disconnection, the broker MUST continue publishing the retained messages as long as the associated tokens are valid.

In case of disconnections due to network errors or server disconnection due to a protocol error (which includes authorization errors), the Will message must be sent if the client supplied a Will in the CONNECT request message. The token provided in the CONNECT request must cover the Will topic. The Will message MUST be published to the Will topic when the network connection is closed regardless of whether the corresponding token has expired.

3. Improved Protocol Interactions with MQTT v5

In the new MQTT v5 - the OASIS Specification Draft [MQTT-OASIS-Standard-v5], several new capabilities are introduced, which enable better integration with the ACE standards. The newly enhanced authentication and re-authentication methods support a much wider range of authentication flows beyond username and password. With the MQTT v5, there is a clearly defined approach for using token-based approaches. Similarly, in MQTT v5, it is possible for a client to request a re-authentication. Finally, MQTT v5 generally improves error reporting, enabling better response to authorization failures during publishing messages to the subscribers.

3.1. Token Transport via Authentication Exchange (AUTH)

To initiate the authentication and authorization flow, as before, the CAS initiates the token request as in Section 2.1. When the client wishes to connect to the RS (broker), it uses the CONNECT message of MQTT. Figure 4 shows the structure of the MQTT CONNECT control message used in MQTT v5.

```
       0            8            16            24            32
       +------------------------------------------------------+
       |CPT=1 | Rsvd.|Remaining len.| Protocol  name len. = 4 |
       +------------------------------------------------------+
       |                      ’M’ ’Q’ ’T’ ’T’                 |
       +------------------------------------------------------+
       | Proto.level=5|Connect flags|          Keep alive     |
       +------------------------------------------------------+
       |                 Property length                      |
       |              Auth. Method (0x15) | ’ace_mqtt_tls’    |
       |          Auth. Data (0x16)   | empty or token        |
       |                                                      |
       +------------------------------------------------------+
```

Figure 4: MQTT CONNECT control message. (CPT=Control Packet Type, Rsvd=Reserved, len.=length, Proto.=Protocol)

To communicate the necessary connection parameters, the client uses the appropriate flags of the CONNECT message. To achieve a clean session (i.e., the session should start without an existing session), the new MQTT v5 session flags MUST be set appropriately: the Clean Start Flag MUST be set to 1 and Session Expiry Interval MUST be set to 0.

With the enhanced authentication capabilities, it is not necessary to overload the username and password fields in the CONNECT message for ACE authentication. Nevertheless, the RS MUST support both methods for supporting the token: (1) Token transport via username and password and (2) using the new AUTH (Authentication Exchange) method. The token transport via username and password is as described in Section 2.1.2. The rest of this section describes the AUTH method.

To use the AUTH method, the username flag MUST be set to 0 and the password flag MUST be set to 0. The client can set the Authentication Method as a property of a CONNECT packet by setting Auth Properties (with the property identifier 0x15). The client must MUST set the UTF-8 encoded string containing the name of the authentication method as ’ace_mqtt_tls’. If the RS does not support this profile, it sends a CONNACK with a Reason Code of ’0x8C (Bad authentication method)’

The Authentication Method is followed by the Authentication Data, which has a property identifier 0x16. Authentication data is binary data and is defined by the authentication method. The RS MAY support different implementations for transporting the authentication data. The first option is that Authentication data contains both the token and the keyed message digest (MAC) or signature as described in Section 2.1.2. In this case, the token validation proceeds as described in Section 2.1.3 and the server responds with a CONNACK. The reason code of the CONNACK ’0x00 (Success)’ if the authentication is successful. In case of an invalid PoP token, the CONNACK reason code is ’0x87 (Not Authorized)’.

The second option that RS may accept is a challenge/response protocol. If the Authentication Data only includes the token, the RS MUST respond with an AUTH packet, with the Authenticate Reason Code set to ’0x18 (Continue Authentication)’. This packet includes the Authentication Method, which MUST be set to ’ace_mqtt_tls’ and Authentication Data. The Authentication Data MUST NOT be empty and contains a challenge for the client. The client responds to this with an AUTH packet, with a reason code ’0x18 (Continue Authentication)’. Similarly, the client packet sets the Authentication Method to ’ace_mqtt_tls’. The Authentication Data in the client’s response contains the signature or MAC computed over the RS’s challenge. To this, the server responds with a CONNACK and a return code of ’0x00 (Success)’ if the authentication is successful. In case of an invalid PoP token, the CONNACK reason code is ’0x87 (Not Authorized)’.

Finally, this document allows the CONNECT message to have an empty Authentication Data field. This is the AS discovery option and the RS responds with a CONNACK reason code ’0x87 (Not Authorized)’ and includes a User Property set to the address of the AS.

3.2. Authorization Errors and Client Re-authentication

MQTT v5 allows better error reporting. To take advantage of this for PUBLISH messages, the QoS level should be set to greater than or equal to 1. This guarantees that RS responds with either a PUBACK or PUBREC packet with a reason code ’0x87 (Not authorized)’ in the case of an authorization error. Similarly, for the SUBSCRIBE case, the SUBACK packet will have a reason code set to ’0x87 (Not authorized)’ for the unauthorized topic(s). When RS is forwarding PUBLISH messages to the subscribed clients, it may discover that some of the subscribers are no more authorized due to expired tokens. In this case, the RS SHOULD send a DISCONNECT message with the reason code ’0x87 (Not authorized)’. Note that the server-side DISCONNECT is a new feature of MQTT v5 (in MQTT v3.1 server needed to drop the connection). RS MUST stop forwarding messages to the unauthorized subscribers.

In the case of a PUBACK with ’0x87 (Not authorized)’, the client can update its token using the Re-authentication feature of MQTT v5. Also, the clients can proactively update their tokens without waiting for such a PUBACK. To re-authenticate, the client sends an AUTH packet with a reason code ’0x19 (Re-authentication)’. The client MUST set the authentication method as ’ace_mqtt_tls’ and transport the new token in the Authentication Data. The client and the RS go through the same steps for proof of possession validation described in the previous section. If the re-authentication fails, the server MUST send a DISCONNECT with the reason code ’0x87 (Not Authorized)’.

4. IANA Considerations

This memo includes no request to IANA.

5. Security Considerations

The security considerations outlined in [I-D.ietf-ace-oauth-authz] apply to this work.

6. Privacy Considerations

The privacy considerations outlined in [I-D.ietf-ace-oauth-authz] apply to this work. Furthermore, the RS is a central trusted party and may forward potentially sensitive information between clients.

7. References

7.1. Normative References

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

[I-D.ietf-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-11 (work in progress), March 2018.

[MQTT-OASIS-Standard]
Banks, A., Ed. and R. Gupta, Ed., "OASIS Standard MQTT Version 3.1.1 Plus Errata 01", 2015, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

[MQTT-OASIS-Standard-v5]
Banks, A., Ed., Briggs, E., Ed., Borgendale, K., Ed., and R. Gupta, Ed., "OASIS Public Review Draft 01 MQTT Version 5.0", 2017, <http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.html>.

[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>.

7.2. Informative References

[fremantle14]
Fremantle, P., Aziz, B., Kopecky, J., and P. Scott, "Federated Identity and Access Management for the Internet of Things", research International Workshop on Secure Internet of Things, September 2014, <http://dx.doi.org/10.1109/SIoT.2014.8>.

[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-06 (work in progress), November 2017.

[I-D.ietf-ace-cwt-proof-of-possession]
Jones, M., Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-02 (work in progress), March 2018.

[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>.

[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>.

Appendix A. Checklist for profile requirements

- AS discovery: For the basic protocol using either MQTT v3.1 or MQTT v5, the clients/client authorization servers need to be configured out-of-band. RS does not provide any hints to help AS discovery. AS discovery is possible with the MQTT v5 extensions described in Section 3.

- The communication protocol between the client and RS: MQTT

- The security protocol between the client and RS: TLS

- Client and RS mutual authentication: RS provides a server certificate during TLS handshake. Client transports token and MAC via the MQTT CONNECT message.

- Content format: For the HTTPS interactions with AS, "application/json". The MQTT payloads may be formatted JSON or CBOR.

- PoP protocols: Either symmetric or asymmetric keys can be supported.

- Unique profile identifier: mqtt_tls

- Token introspection: RS uses HTTPS /introspect interface of AS.

- Token request: CAS uses HTTPS /token interface of AS.

- /authz-info endpoint: It MAY be supported using the method described in Appendix B, not protected.

- Token transport: In MQTT CONNECT message or using the AUTH extensions for MQTT v5 described in Section 3.

Appendix B. The authorization information endpoint

The main document described a method for transporting tokens inside MQTT CONNECT messages. In this section, we describe an alternative method to transport an access token.

The method consists of the MQTT broker accepting PUBLISH messages to a public "authz-info" topic. A client using this method MUST first connect to the broker, and publish the access token using the "authz-info" topic. The broker must verify the validity of the token (i.e., through local validation or introspection). After publishing the token, the client disconnects from the broker and is expected to try reconnecting over TLS.

In MQTT v3.1, after the client published to the "authz-info" topic, it is not possible for the broker to communicate the result of the token verification. In MQTT v5, the broker can return 'Not authorized' error to a PUBLISH request for QoS greater or equal to 1. In any case, any token authorization failure will affect the TLS handshake, which can prompt the client to obtain a valid token.

Appendix C. Document Updates

Version 01 updates Version 00 as follows:

- Adds Section 3 to describe improvements to the basic protocol operation with the new MQTT v5 - OASIS Specification Draft [MQTT-OASIS-Standard-v5], including improved authentication exchange and error reporting.

- Condenses background information specific to MQTT in Section 2.

- Clarifies token transport and token structure in Section 2.1.2 and Section 2.1.3.

- Removes Appendix on error reporting as this is now handled with MQTT v5.

Version 02 updates Version 01 as follows:

- Adds PINGREQ packet for token expiry checks.

- Minor typo fixes.

Acknowledgements

The authors would like to thank Ludwig Seitz for his input on the authorization information endpoint, presented in the appendix.

Authors’ Addresses

Cigdem Sengul
Nominet
2 Kingdom Street
London W2 6BD
UK

Email: Cigdem.Sengul@nominet.uk

Anthony Kirby
Nominet
Minerva House, Edmund Halley Road
Oxford OX4 4DQ
UK

Email: Anthony.Kirby@nominet.uk

Paul Fremantle
University of Portsmouth
School of Computing, Buckingham House
Portsmouth PO1 3HE
UK

Email: paul.fremantle@port.ac.uk

ACE Working Group   M. Tiloca
Internet-Draft   RISE SICS AB
Intended status: Standards Track   J. Park
Expires: September 6, 2018   Universitaet Duisburg-Essen
                                                          March 05, 2018

Joining OSCORE groups in ACE
draft-tiloca-ace-oscoap-joining-03

Abstract

This document describes a method to join a group 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 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

Copyright Notice

Table of Contents

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 also to protect CoAP group communication over IP multicast [RFC7390]. This relies on a Group Manager entity, which is responsible for managing an OSCORE group, where members exchange CoAP messages secured with OSCORE. In particular, the Group Manager coordinates the join process of new group members and can be responsible for multiple groups.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines how a client joins an OSCORE group through a resource server acting as Group Manager. The client acting as joining node relies on an Access Token, which is bound to a proof-of-possession key and authorizes the access to 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 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 [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile], as well as 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]. Message exchanges are presented as RESTful protocol interactions, for which HTTP [RFC7231] provides useful terminology.

The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors]. 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].

This document refers also to the following terminology.

- 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].

- 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.

- 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.

- Join endpoint: an endpoint at the Group Manager associated to a join resource.

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 explicitly 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 a network node joins an OSCORE group by using the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- 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 group.

- The joining node acts as Client (C), and requests to join an OSCORE group by accessing the related join endpoint at the Group Manager.

- The Authorization Server (AS) enables and enforces the authorized access of joining nodes to join endpoints at the Group Manager, and hence the access to the related OSCORE groups. Multiple Group Managers can be associated to the same AS. The AS is not necessarily expected to release Access Tokens for any other purpose than accessing join resources on registered Group Managers. However, the AS may be configured also to release Access Tokens for accessing resources at members of OSCORE groups.

All communications between the involved entities rely on the CoAP protocol and must be secured. 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].

Communications between the joining node and the AS (/token endpoint) as well as between the Group Manager and the AS (/introspection endpoint) can be secured by different means, for instance by means of 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.

The following steps are performed for joining 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) referred in [I-D.palombini-ace-key-groupcomm].

1. The joining node requests an Access Token from the AS to access a join resource on the Group Manager and hence the associated OSCORE group (see Section 3). The response from the AS enables the joining node to start a secure channel with the Group Manager, if not already established.

2. The joining node transfers authentication and authorization information to the Group Manager by posting the obtained Access Token. Then, the joining node and the Group Manager have to establish a secure channel in case one is not already set up (see Section 4). That is, a joining node must establish a secure communication channel with a Group Manager, before joining an OSCORE group under that Group Manager for the first time.

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 in the OSCORE group.

3. Joining Node to Authorization Server

This section considers a joining node that intends to contact the Group Manager for the first time. That is, the joining node has never attempted before to join an OSCORE group under that Group Manager. Also, the joining node and the Group Manager do not have a secure communication channel established.

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 2 of [I-D.ietf-ace-dtls-authorize] to discover the correct AS in charge of the Group Manager. As an alternative, the joining node may look up in a Resource Directory service [I-D.ietf-core-resource-directory].

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:

- The "scope" parameter MUST be present and includes:

  * in the first element, the Group Identifier (Gid) of the group to join under the Group Manager. This identifier may not fully coincide with the Gid currently associated to the respective group, e.g. if it includes a dynamic component.

  * in the second element, which MUST be present, the role(s) that the joining node intends to have in the group it intends to join. Roles and their combinations are defined in [I-D.ietf-core-oscore-groupcomm], and indicated as "multicaster", "listener" and "purelistener". Multiple roles are specified in the form of a CBOR array.

- The "aud" parameter MUST be present and is set to the address of the Group Manager.

- The "get_pub_keys" parameter is present only if the Group Manager is configured to store the public keys of the group members and, at the same time, the joining node wants to retrieve such public keys during the joining process (see Section 5). In any other case, this parameter MUST NOT be present.

3.2. Authorization Response

The AS is responsible for authorizing the joining node, accordingly to group join policies enforced on behalf of the 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, since defining the lifetime of Access Tokens is out of the scope of this specification.

In case the value of "scope" specified in the Access Token differs from the value originally included in the Access Token request, the Access Token response MUST include the "scope" parameter, whose second element 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 5.6.4.4 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 "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 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 of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The "get_pub_keys" parameter can be present only if included also in the Authorization Request previously sent to the AS. In such a case, its value is the same as in the Authorization Request. Otherwise, this parameter MUST NOT be present.

- The "client_cred" parameter, if present, includes the public key or certificate of the joining node. Specifically, it includes the public key of the joining node if the Group Manager is configured to store the public keys of the group members, or the certificate of the joining node otherwise. 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.

- The "pub_keys_repos" parameter MAY be present if the "client_cred" parameter is both present and with value a certificate of the joining node. If present, this parameter contains the list of public key repositories storing the certificate of the joining node. In any other case, this parameter MUST NOT be present.

4.2. Join Response

The Group Manager processes the request according to [I-D.ietf-ace-oauth-authz]. If this yields to a positive outcome, the Group Manager updates the group membership by registering the joining node as a new member of the OSCORE group.

Then, the Group Manager replies to the joining node providing the information necessary to participate in the group communication. This join response follows the format of the Key Distribution success Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The "key" parameter includes what the joining node needs in order to set up the OSCORE Security Context as in 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 "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 MUST be present and has as value the OSCORE Endpoint ID assigned to the joining node by the Group Manager.

  * The "serverID" parameter MUST be present and has as value the Group Identifier (Gid) currently 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.

- 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 multicaster; and ii) the public keys of the multicasters currently in the group, in case the joining node is configured (also) as listener or pure listener.

- 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]), as well as the rekeying protocol used to renew the keying material in the group (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]).

- The "mgt_key_material" parameter SHOULD be present and includes the administrative keying material that the joining node requires to participate in the rekeying process led by the Group Manager. The exact content and format depend on the specific rekeying protocol used in the group.

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 Section 4 of [I-D.ietf-core-oscore-groupcomm].

5. Public Keys of Joining Nodes

Source authentication of OSCORE messages exchanged within the group is ensured by means of digital counter signatures [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 the source authenticity of incoming group messages.

Upon joining an OSCORE group, a joining node is expected to make its own public key available to the other group members, either through the Group Manager or through another trusted, publicly available, key repository. However, this is not required for a node that joins a group exclusively as pure listener.

As also discussed in Section 6 of [I-D.ietf-core-oscore-groupcomm], it is recommended that the Group Manager is configured to store the public keys of the group members and to provide them upon request. If so, three cases can occur when a new node joins a group.

- The Group Manager already acquired the public key of the joining node during a previous join process. In this case, the joining node is not required to provide again its own public key to the Group Manager.

- 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.

- 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).

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.

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).

On the other hand, in case the Group Manager is not configured to store public keys of group members, the joining node provides the Group Manager with its own certificate in the "client_cred" parameter of the join request targeting the join endpoint (see Section 4.1). Then, the Group Manager validates and handles the certificate, for instance as described in Appendix D.2 of [I-D.ietf-core-oscore-groupcomm].

6. 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.

- Management of group keying material (Section 2.1). This includes the need to revoke and renew the keying material currently used in the OSCORE group, upon changes in the group membership. In particular, renewing the keying material is required upon a new node joining the group, in order to preserve backward security. That is, the Group Manager should renew the keying material before completing the join process and sending a join response. Such a join response provides the joining node with updated the keying material just established in the group. The Group Manager is responsible to enforce rekeying policies and accordingly update the keying material in the groups of its competence (Section 6).

- Synchronization of sequence numbers (Section 5). This concerns how a listener node that has just joined an OSCORE group can synchronize with the sequence number of multicasters in the same group.

- Provisioning and retrieval of public keys (Appendix D.2). This provides guidelines about how to ensure the availability of group members' public keys, possibly relying on the Group Manager as trusted key repository (Section 6).

Further security considerations are inherited from the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], as well as from the specific profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Martin Gunnarsson, Francesca Palombini, Jim Schaad, Ludwig Seitz and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE.

9. References

9.1. Normative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profiles for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-02 (work in progress), October 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-10 (work in progress), February 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-00 (work in progress), December 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-08 (work in progress), January 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-01 (work in progress), March 2018.

[I-D.palombini-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-palombini-ace-key-groupcomm-00 (work in progress), March 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>.

9.2. 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-06 (work in progress), November 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-12 (work in progress), October 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>.

[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>.

[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>.

Authors’ Addresses

Marco Tiloca
RISE SICS 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

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 will expire on July 26, 2018.

Copyright Notice

Table of Contents

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.

- 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

- DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA); this could be a common case for simple re-enrollment of clients;

- DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

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.

```
+------------------------------------------------+
|    EST request/response messages               |
+------------------------------------------------+
|    CoAP for message transfer and signaling     |
+------------------------------------------------+
|    DTLS for transport security                 |
+------------------------------------------------+
|    UDP for transport                           |
+------------------------------------------------+
```

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:

- Simple enroll and reenroll, for CA to sign public client-identity key.

- CA certificate retrieval, needed to receive the complete set of CA certificates.

- CSR Attributes request messages, informs the client of the fields to include in generated CSR.

- 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:

- A proxy between any EST-client and EST-server

```
                                        Constrained Network
                     .---------.    .----------------------------.
                     |   RA    |    |.--------------------------.|
                     '---------'    ||                          ||
                           |        ||                          ||
   .------.  HTTP   .-----------------.  CoAPS   .-----------.  ||
   | EST  |<------->|ESTcoaps-to-HTTPS|<-------->| EST Client|  ||
   |Server|over TLS |      Proxy      |          '-----------'  ||
   '------'         '-----------------'                         ||
                                    ||                          ||
                                    |'--------------------------'|
                                    '----------------------------'
```

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 then 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 =
30233906092a6206734107028c2a302326020101310030 0b06092a6206734107018
c0c3020bb302063c20102020900a61e75193b7acc0d06092a620673410105050030
1b311930170603550403131065737444578616d706c654341204f774f301e170d313
33035303930333533333315a170d31343035303930333533333315a301b3119301706
03550403131065737444578616d706c654341204f774f302062300d06092a6206734
10101050003204f0030204a022041003a923a2968bae4aae136ca4e2512c5200680
358482ac39d6f640e4574e654ea35f48b1e054c5da3372872f7a1e429f4edf39584
32efb2106591d3eb783c1034709f251fc86566bda2d541c792389eac4ec9e181f4b
9f596e5ef2679cc321542b11337f90a44df3c85f1516561fa968a1914f265bc0b82
76ebe3106a790d97d34c8c37c74fe1c30b396424664ac426284a9f6022e02693843
6880adfcd95c98ca1dfc2e6d75319b85d0458de28a9d13fb16d620fff7541f6a25d
7daf004355020301000130b040300f0603551d130101f10530030101fc1d0603551
d0e04160414084d321ca0135e77217a486b686b334b00e0603551d0f0101f104030
```

```
   20106300d06092a62067341010505000320410023703b965746a0c2c978666d787a
   94f89b495a11f0d369b28936ec2475c0f0855c8e83f823f2b871a1d92282f323c45
   904ba008579216cf5223b8b1bc425a0677262047f7700240631c17f3035d1c3780b
   2385241cba1f4a6e98e6be6820306b3a786de5a557795d1893822347b5f825d34a7
   ad2876f8feba4d525b31066f6505796f71530003431a3e6bbfe788b4565029a7e20
   a51107677552586152d051e8eebf383e92288983421d5c5652a4870c3af74b9bdbe
   d6b462e2263d30f6d3020c330206bc2010202010130 0d06092a620673410105050 0
   301b311930170603550403131065737445786 16d706c654341204f774f301e170d3
   13330353039303335333332 5a170d31343035303930333533333325a301b31193017
   0603550403131065737445786 16d706c654341204e774f302062300d06092a62067
   3410101050003204f0030204a02204100ef6b677a3247c1fc03d2b9baf113e5e7e1
   13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
   9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
   7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
   c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
   768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
   6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
   530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
   551d0f0101f10403020106300d06092a620673410105050003204100423f06d4b76
   0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
   475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
   e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
   34119794b90e320bc730d6c1bee496e7ac125ce9a1eca595a3a4c54a865e6b623c9
   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)
30208530206d020100301f311d301b06035504031314656d6f7374657034203
1333638313431333532302062300d06092a620673410101050003204f0030204a
022041005d9f4dffd3c5949f646a9584367778560950b355c35b8e34726dd3764
54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
b871691e0f02030100018701f06092a62067341090731121310 2b72724369722f
372b45597535305434300d06092a620673410105050003204100441b40177a3a6
5501487735a8ad5d3827a4eaa867013920e2afcda87aa81733c7c0353be47e1bf
a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
17071103407553155cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40

RET:
2.05 Content (Content-Format: application/pkcs7-mime)
3020f806092a6206734107028329302 0e50201013100300b06092a62067341070
1830b3020c730206fc20102020115300d06092a6206734101050500301b311930
1706035504031310657374457861 6d706c654341204e774e301e170d313330353
0393233313535335a170d31343035303932333135353 35a301f311d301b060355
040313146465 6d6f73746570342031333638313431333532302062300d06092a6
20673410101050003204f0030204a022041005d9f4dffd3c5949f646a95843677
78560950b355c35b8e34726dd376454231734795b4c09b9c6d75d408311307a81
f7adef7f5d241f7d5be85620c5d4438bbb4242cf215c167f2ccf36c364ea2618a
62f0536576369d6304e6a968772247d86824f079faac7a6f694cfda5b84c42087
dc062d462190c525813f210a036a737b4f30d8891f4b75559fb72752453146332
d51c937557716ccec624f5125c3a4447ad3115020048113fef54ad554ee88af09
a2583aac9024075113db4990b1786b871691e0f020301000134b050300e060355
1d0f0101f104030204c1d0603551d0e04160414e81d0788aa2710304c5ecd4d1e
065701f0603551d230418301653112966e304761732fbfe6a2c823c300d06092a
62067341010505000320410029 10d86f2ffeeb914c046816871de601567d291b4
3fabee0f0e8ff81cea27302a7133e20e9d04029866a8963c7d14e26fbe8a0ab1b
77fbb1214bbcdc906fbc381137ec1de685f79406c3e416b8d82f97174bc691637
5a4e1c4bf744c7572b4b2c6bade9fb35da786392ee0d95e3970542565f3886ad6
7746d1b12484bb02616e63302dc371dc6006e431fb7c457598dd204b367b0b3d3
258760a303f1102db26327f929b7c5a60173e1799491b6915024875602 6b80553
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 incudes 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
3020813020690201003005b313e303c0603550403133357365727665724b6579476
56e2072657120627920636c69656e7420696e2064656d6f207374657020313220
3133363831343139353531193017060355040513105049443a576964676574205
34e3a3130302062300d06092a620673410101050003204f0030204a02204100f4
dfa6c03f7f2766b23776c333d2c0f9d1a7a6ee36d01499bbe6f075d1e38a57e98
ecc197f51b75228454b7f19652332de5e52e4a974c6ae34e1df80b33f15f47d3b
cbf76116bb0e4d3e04a9651218a476a13fc186c2a255e4065ff7c271cff104e47
31fad53c22b21a1e5138bf9ad0187314ac39445949a48805392390e78c7659621
6d3e61327a534f5ea7721d2b1343c7362b37da502717cfc2475653c7a3860c5f4
0612a5db6d33794d755264b6327e3a3263b149628585b85e57e42f6b3277591b0
2030100018701f06092a620673410907311213106446734158 6d4a6e6a6f6b427
4447672300d06092a620673410105050003204100472d11007e5a2b2c2023d47a
6d71d046c307701d8ebc9e47272713378390b4ee321462a3dbe54579f5a514f6f
4050af497f428189b63655d03a194ef729f101743e5d03fbc6ae1e84486d1300a
f9288724381909188c851fa9a5059802eb64449f2a3c9e441353d136768da27ff
4f277651d676a6a7e51931b08f56135a2230891fd184960e1313e7a1a9139ed19
28196867079a456cd2266cb754a45151b7b1b939e381be333fea61580fe5d25bf
4823dbd2d6a98445b46305c10637e202856611
```

```
RET:
2.05 Content (Content-Format: application/pkcs8)
30213e020100300d06092a620673410101050004212830212402010002204100 3
c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274
dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1
1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c
0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7
```

```
   f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
   542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
   1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
   ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
   1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
   33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
   8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
   0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
   c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef
   7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
   83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
   241366b7907187583b511102103369047e5cce0b65012200df5ec697b5827575c
   db6821ff299d6a69574b31ddf0fbe9245ea2f74396c24b3a7565067e41366423b
   5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
   173de2c3aa1f200af535411c7090210401421e2ea217e37312dcc606f453a6634
   f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
   50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
   f917438f607f18181684376e13a39e07
   --estServerExampleBoundary
   3020c506092a6206734107028336302 0f20201013100300b06092a62067341070
   183183020d430207cc20102020116300d06092a62067341010505000301b311930
   17060355040313106573744578616d706c654341204e774e301e170d313330353
   03932333235353 65a170d3134303530393233323535365a302c312a3028060355
   04031321736572766572736966465206b65792067656e65726174656420726573 7
   06f6e7365302062300d06092a62067341010105000 3204f0030204a022041003c
   0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274d
   d01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a11
   bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c0
   c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d54
   5e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704e
   fec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e310830100
   0134b050300e0603551d0f0101f104030204c1d0603551d0e04160414764b1bd5
   e69935626e476b195a1a8c1f0603551d230418301653112966e304761732fbfe6
   a2c823c300d06092a62067341010505000320410 0474e5100a9cdaaa813b30f48
   40340fb17e7d6d6063064a5a7f2162301c464b5a8176623dfb1a4a484e618de1c
   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 =
   30233906092a6206734107028c2a302326020101310030b06092a6206734107018
   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
   1b3119301706035504031310657374445786 16d706c654341204f774f301e170d313
   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