ACE
Internet-Draft
Intended status: Standards Track
Expires: August 10, 2019

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
M. Richardson
SSW
S. Raza
RISE SICS

February 6, 2019

EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-08

Abstract

Enrollment over Secure Transport (EST) is used as a certificate provisioning protocol over HTTPS. Low-resource devices often use the lightweight Constrained Application Protocol (CoAP) for message exchanges. This document defines how to transport EST payloads over secure CoAP (EST-coaps), which allows constrained devices to use existing EST functionality for provisioning certificates.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 10, 2019.

Table of Contents

1. Change Log

EDNOTE: Remove this section before publication

-08

added application/pkix-cert Content-Format TBD287.

Stated that well-known/est is compulsory

Use of response codes clarified.

removed bugs: Max-Age and Content-Format Options in Request

Accept Option explained for est/skg and added in enroll example

Persistenc of DTLS connection clarified.

Minor text fixes.

-07:

redone examples from scratch with openssl

Updated authors.

Added CoAP RST as a MAY for an equivalent to an HTTP 204 message.

Added serialization example of the /skg CBOR response.

Added text regarding expired IDevIDs and persistent DTLS connection that will start using the Explicit TA Database in the new DTLS connection.

Nits and fixes

Removed CBOR envelop for binary data

Replaced TBD8 with 62.

Added RFC8174 reference and text.

Clarified MTI for server-side key generation and Content-Formats. Defined the /skg MTI (PKCS#8) and the cases where CMS encryption will be used.

Moved Fragmentation section up because it was referenced in sections above it.

-06:

clarified discovery section, by specifying that no discovery may be needed for /.well-known/est URI.

added resource type values for IANA

added list of compulsory to implement and optional functions.

Fixed issues pointed out by the idnits tool.

Updated CoAP response codes section with more mappings between EST HTTP codes and EST-coaps CoAP codes.

Minor updates to the MTI EST Functions section.

Moved Change Log section higher.

-05:

repaired again

TBD8 = 62 removed from C-F registration, to be done in CT draft.

-04:

Updated Delayed response section to reflect short and long delay options.

-03:

Removed observe and simplified long waits

Repaired Content-Format specification

-02:

Added parameter discussion in section 8

Concluded Content-Format specification using multipart-ct draft

examples updated

-01:

Editorials done.

Redefinition of proxy to Registrar in Section 7. Explained better the role of https-coaps Registrar, instead of "proxy"

Provide "observe" Option examples

extended block message example.

inserted new server key generation text in Section 5.8 and motivated server key generation.

Broke down details for DTLS 1.3

New Media-Type uses CBOR array for multiple Content-Format payloads

provided new Content-Format tables

new media format for IANA

-00

copied from vanderstok-ace-coap-04

2. Introduction

"Classical" 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). EST transports messages over HTTPS.

This document defines a new transport for EST based on the Constrained Application Protocol (CoAP) since some Internet of Things (IoT) devices use CoAP instead of HTTP. Therefore, this specification utilizes DTLS [RFC6347], CoAP [RFC7252] and UDP instead of TLS [RFC8446], HTTP [RFC7230] and TCP.

EST responses can be relatively large and for this reason this specification also uses CoAP Block-Wise Transfer [RFC7959] to offer a fragmentation mechanism of EST messages at the CoAP layer.

This document also profiles the use of EST to only support certificate-based client authentication. HTTP Basic or Digest authentication (as described in Section 3.2.3 of [RFC7030] are not supported.

3. 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.

Many of the concepts in this document are taken from [RFC7030]. Consequently, much text is directly traceable to [RFC7030].

4. Conformance to RFC7925 profiles

This section describes how EST-coaps fits into the profiles of low-resource devices described in [RFC7925]. EST-coaps can transport certificates and private keys. Certificates are responses to (re-)enrollment requests or requests for a trusted certificate list. Private keys can be transported as responses to a server-side key generation request as described in section 4.4 of [RFC7030] and discussed in Section 5.8 of this document.

As per Sections 3.3 and 4.4 of [RFC7925], the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. Curve secp256r1 MUST be supported [RFC8422]; this curve is equivalent to the NIST P-256 curve. Crypto agility is important, and the recommendations in [RFC7925] section 4.4 and any updates to RFC7925 concerning Curve25519 and other CFRG curves also apply.

DTLS1.2 implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC8422]. Uncompressed point format MUST also be supported. [RFC6090] is a summary of the ECC algorithms. DTLS 1.3 [I-D.ietf-tls-dtls13] implementations differ from DTLS 1.2 because they do not support point format negotiation in favor of a single point format for each curve. Thus, support for DTLS 1.3 does not mandate point formation extensions and negotiation.

The authentication of the EST-coaps server by the EST-coaps client is based on certificate authentication in the DTLS handshake. The EST-coaps client MUST be configured with at least an Implicit TA database from its manufacturer which will enable the authentication of the server the first time before updating its trust anchor (Explicit TA) [RFC7030].

The authentication of the EST-coaps client MUST be with a client certificate in the DTLS handshake. This can either be

- 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.

- a previously installed certificate (e.g., manufacturer IDevID [ieee802.1ar] or a certificate issued by some other party); the server is expected to trust that certificate. IDevID's are expected to have a very long life, as long as the device, but under some conditions could expire. In that case, the server MAY want to authenticate a client certificate against its trust store although the certificate is expired (Section 11).

5. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to avoid (excessive) fragmentation of UDP datagrams. The use of Blocks for the transfer of larger EST messages is specified in Section 5.6. Figure 1 below shows the layered EST-coaps architecture.

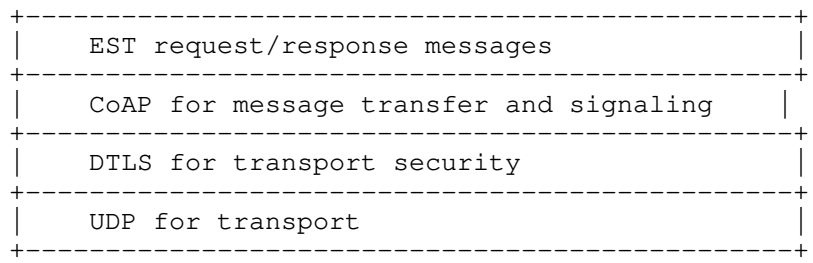


Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design. The supported message types in EST-coaps are:

- CA certificate retrieval, needed to receive the complete set of CA certificates.

- Simple enroll and reenroll, for a CA to sign public client-identity key.

- Certificate Signing Request (CSR) Attributes messages that 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 choses so.

5.1. Discovery and URIs

EST-coaps is targeted for low-resource networks with small packets. Saving header space is important and short EST-coaps URIs are specified in this document. These URIs are shorter than the ones in [RFC7030]. Two example EST-coaps resource path names are:

coaps://est-coaps.example.ietf.org:<port>/.well-known/est/<short-est>
coaps://est-coaps.example.ietf.org:<port>/.well-known/est/
ArbitraryLabel/<short-est>

The short-est strings are defined in Table 1. The ArbitraryLabel path-segment, if used, SHOULD be of the shortest length possible (Sections 3.1 and 3.2.2 of [RFC7030]. Arbitrary Labels are usually defined and used by EST CAs in order to route client requests to the appropriate certificate profile.

The EST-coaps server URIs, obtained through discovery of the EST-coaps root resource(s) as shown below, are of the form:

coaps://est-coaps.example.ietf.org:<port>/<root-resource>/<short-est>
coaps://est-coaps.example.ietf.org:<port>/<root-resource>/
ArbitraryLabel/<short-est>

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: Table 1: Short EST-coaps URI path

Clients and servers MUST support the short resource URIs. The corresponding longer URIs from [RFC7030] MAY be supported.

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. The example below shows the

discovery of the presence and location of EST-coaps resources. Linefeeds are included only for readability.

```
  REQ: GET /.well-known/core?rt=ace.est*

  RES: 2.05 Content
</est>; rt="ace.est",
</est/crts>;rt="ace.est.crts";ct="281 TBD287",
</est/sen>;rt="ace.est.sen";ct="281 TBD287",
</est/sren>;rt="ace.est.sren";ct="281 TBD287",
</est/att>;rt="ace.est.att";ct=285,
</est/skg>;rt="ace.est.skg";ct="62 280 284 281 TBD287"
```

The first line of the discovery response above MUST be included. The five consecutive lines after the first MAY be included. The return of the content types allows the client to choose the most appropriate one.

Port numbers, not returned in the example, are assumed to be the default numbers 5683 and 5684 for CoAP and CoAPS respectively (Sections 12.6 and 12.7 of [RFC7252]). Discoverable port numbers MAY be returned in the <href> of the payload [I-D.ietf-core-resource-directory]. An example response payload for non-default CoAPS server port 61617 follows below. Linefeeds were included only for readability.

```
 REQ: GET /.well-known/core?rt=ace.est*

 RES: 2.05 Content
<coap://[2001:db8:3::123]:61617/est>;rt="ace.est";
                              anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/crts>;rt="ace.est.crts";
             ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sen>;rt="ace.est.sen";
             ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/sren>;rt="ace.est.sren";
             ct="281 TBD287";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/att>;rt="ace.est.att";
                    ct="285";anchor="coap://[2001:db8:3::123]:61617",
<coap://[2001:db8:3::123]:61617/est/skg>;rt="ace.est.skg";
    ct="62 280 284 281 TBD287";anchor="coap://[2001:db8:3::123]:61617"
```

The server MUST support the default /.well-known/est server root resource and port 5684. Resource discovery is necessary when the IP address of the server is unknown to the client. Resource discovery SHOULD be employed when non-default URIs (like /est or /est/ ArbitraryLabel) or ports are supported by the server, when the client is unaware of what EST-coaps resources are available or if the client

considers sending two Uri-Path Options to convey the resource is wasteful.

It is up to the implementation to choose its root resource; throughout this document the example root resource /est is used.

5.2. Mandatory/optional EST Functions

This specification contains a set of required-to-implement functions, optional functions, and not specified functions. The latter ones are deemed too expensive for low-resource devices in payload and calculation times.

Table 2 specifies the mandatory-to-implement or optional implementation of the est-coaps functions.

EST Functions	EST-coaps implementation
/cacerts	MUST
/simpleenroll	MUST
/simplereenroll	MUST
/fullcmc	Not specified
/serverkeygen	OPTIONAL
/csrattrs	OPTIONAL

Table 2: Table 2: List of EST-coaps functions

While [RFC7030] permits a number of these functions to be used without authentication, this specification requires that the client MUST be authenticated for all functions.

5.3. Payload formats

The Content-Format (HTTP 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 (section 3.2.2 of [RFC7030]) are in EST-coaps specified by the Content-Format Option (12) of CoAP. The combination of URI-Path and Content-Format in EST-coaps MUST map to an allowed combination of URI and Media-Type in EST. The required Content-Formats for these requests and response messages are defined in Section 10.1. The CoAP response codes are defined in Section 5.5.

Content-Format TBD287 can be used in place of 281 to carry a single certificate instead of a PKCS#7 container in a /crts, /sen, /sren or /skg response. Content-Format 281 MUST be supported by EST-coaps

servers. Servers MAY also support Content-Format TBD287. It is up to the client to support only Content-Format 281, TBD287 or both. The client is expected to use an COAP Accept Option in the request to express the preferred response Content-Format. If an Accept Option is not included in the request, the client is not expressing any preference and the server SHOULD choose format 281. If the preferred Content-Format cannot be returned, the server MUST send a 4.06 (Not Acceptable) response, unless another error code takes precedence for the response [RFC7252].

Content-Format 286 is used in /sen, /sren and /skg requests and 285 in /att responses.

EST-coaps is designed for low-resource devices and hence does not need to send Base64-encoded data. Simple binary is more efficient (30% smaller payload) and well supported by CoAP. Thus, the payload for a given Media-Type follows the ASN.1 structure of the Media-Type and is transported in binary format.

application/multipart-core

A representation with Content-Format identifier 62 contains a collection of representations along with their respective Content-Format. The Content-Format identifies the Media-Type application/multipart-core specified in [I-D.ietf-core-multipart-ct].

The collection is encoded as a CBOR array [RFC7049] with an even number of elements. The second, fourth, sixth, etc. element is a binary string containing a representation. The first, third, fifth, etc. element is an unsigned integer specifying the Content-Format identifier of the consecutive representation. For example, a collection containing two representations in response to a EST-coaps server-side key generation request, could include a private key in PKCS#8 [RFC5958] with Content-Format identifier 284 (0x011C) and a single certificate in a PKCS#7 container with Content-Format identifier 281 (0x0119). Such a collection would look like [284,h'0123456789abcdef', 281,h'fedcba9876543210'] in diagnostic CBOR notation. The serialization of such CBOR content would be

```
84                  # array(4)
19 011C             # unsigned(284)
48                  # bytes(8)
   0123456789ABCDEF # "\x01#Eg\x89\xAB\xCD\xEF"
19 0119             # unsigned(281)
48                  # bytes(8)
   FEDCBA9876543210 # "\xFE\xDC\xBA\x98vT2\x10"
```

Multipart /skg response serialization

When the returned certificate is a single X.509 certificate (not a PKCS#7 container) the Content-Format identifier is TBD287 (0x011F) instead of 281. In cases where the private key is encrypted with CMS (as explained in Section 5.8) the Content-Format identifier is 280 (0x0118) instead of 284. The key and certificate representations are ASN.1 encoded in binary format. An example is shown in Appendix A.3.

5.4. Message Bindings

The general EST-coaps message characteristics are:

- All EST-coaps messages expect a response from the server, thus the client MUST send the requests over confirmable CON CoAP messages.

- The Ver, TKL, Token, and Message ID values of the CoAP header are not affected by EST.

- The CoAP Options used are Uri-Host, Uri-Path, Uri-Port, Content-Format, Accept and Location-Path. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages. The Uri-Host and Uri-Port Options are optional. They are usually omitted as the DTLS destination and port are sufficient. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

- EST URLs are HTTPS based (https://), in CoAP these are assumed to be translated to CoAPS (coaps://)

Table 1 provides the mapping from the EST URI path to the EST-coaps URI path. Appendix A includes some practical examples of EST messages translated to CoAP.

5.5. CoAP response codes

Section 5.9 of [RFC7252] and Section 7 of [RFC8075] specify the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request (/cacerts, /csrattrs), in EST-coaps the equivalent CoAP response code 2.05 or 2.03 MUST be used. Similarly, 2.01, 2.02 or 2.04 MUST be used in response to EST POST requests (/simpleenroll, /simplereenroll, /serverkeygen).

Response code HTTP 202 Retry-After that existed in EST has no equivalent in CoAP. Retry-After is used in EST for delayed server responses. Section 5.7 specifies how EST-coaps handles delayed messages.

EST makes use of HTTP 204 and 404 responses when a resource is not available for the client. The equivalent CoAP codes to use in an EST-coaps responses are 2.04 and 4.04. Additionally, EST's HTTP 401 error translates to 4.01 in EST-coaps. Other EST HTTP error messages are 400, 423 and 503. Their equivalent CoAP errors are 4.00, 4.03 and 5.03 respectively. In case a CoAP Option is unrecognized and critical, the server is expected to return a 4.02 (Bad Option). Moreover, if the Content-Format requested in the client Accept Option, is not supported the server MUST return a 4.06 (Not Acceptable), unless another error code takes precedence for the response.

5.6. Message fragmentation

DTLS defines fragmentation only for the handshake 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 [ieee802.15.4] 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 in EST-coaps. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and Object Identifier (OID) fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, Subject Alternative Names (SAN) and cert fields. For 384-bit curves, ECDSA certificates increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certificates 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. Even with ECC, EST-coaps messages can still exceed MTU sizes on the Internet or 6LoWPAN [RFC4919] (Section 2 of [RFC7959]). EST-coaps needs to be able to fragment messages into multiple DTLS datagrams.

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. As explained in Section 1 of [RFC7959], block-wise transfers should be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks. The EST-coaps client and server MUST support Block2. Block1 MUST be supported for EST-coaps enrollment requests that exceed the Path MTU.

[RFC7959] also defines Size1 and Size2 Options to provide size information about the resource representation in a request and response. EST-client and server MAY support Size1 and Size2 Options.

Examples of fragmented EST-coaps messages are shown in Appendix B.

5.7. Delayed Responses

Server responses can sometimes be delayed. According to section 5.2.2 of [RFC7252], a slow server can acknowledge the request and respond later with the requested resource representation. In particular, a slow server can respond to an EST-coaps enrollment request with an empty ACK with code 0.00, before sending the certificate to the server after a short delay. If the certificate response is large, the server will need more than one Block2 blocks to transfer it. This situation is shown in Figure 2. The client sends an enrollment request that uses N1+1 Block1 blocks. The server uses an empty 0.00 ACK to announce the delayed response which is provided later with 2.04 messages containing N2+1 Block2 Options. The first 2.04 is a confirmable message that is acknowledged by the client with an ACK. Onwards, having received the first 256 bytes in the first Block2 block, the client asks for a block reduction to 128 bytes in a confirmable enrollment request Uri-Path and acknowledges the Block2 blocks sent up to that point.

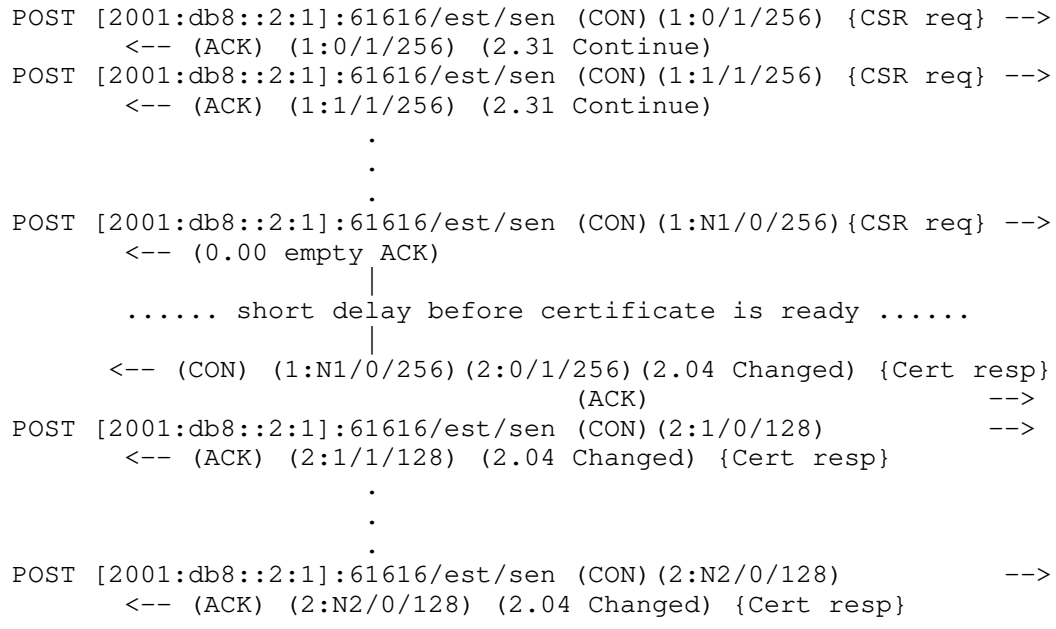


Figure 2: EST-COAP enrollment with short wait

If the server is very slow (i.e. minutes) in providing the response (i.e. when a manual intervention is needed), the server SHOULD respond with an ACK containing response code 5.03 (Service unavailable) and a Max-Age Option to indicate the time the client SHOULD wait to request the content later. After a delay of Max-Age, the client SHOULD resend the identical CSR to the server. As long as the server responds with response code 5.03 (Service Unavailable) with a Max-Age Option, the client SHOULD keep resending the enrollment request until the server responds with the certificate or the client abandons for other reasons.

To demonstrate this scenario, Figure 3 shows a client sending an enrollment request that uses N1+1 Block1 blocks to send the CSR to the server. The server needs N2+1 Block2 blocks to respond, but also needs to take a long delay (minutes) to provide the response. Consequently, the server uses a 5.03 ACK response with a Max-Age Option. The client waits for a period of Max-Age as many times as he receives the same 5.03 response and retransmits the enrollment request until he receives a certificate in a fragmented 2.01 response. Note that the server asks for a decrease in the block size when acknowledging the first Block2.

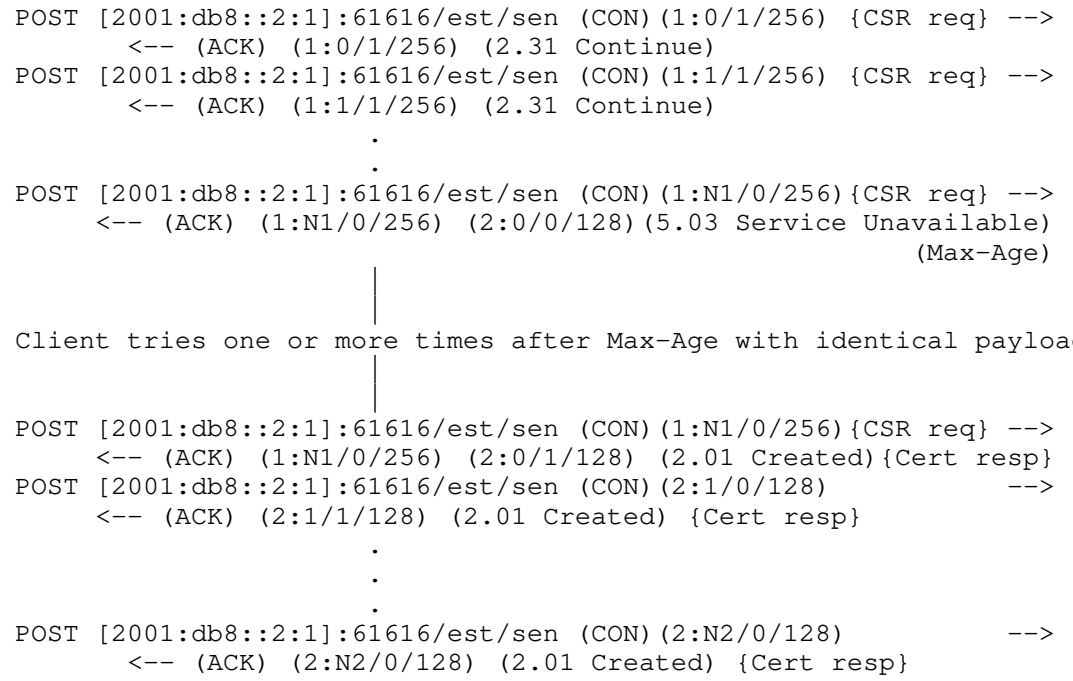


Figure 3: EST-COAP enrollment with long wait

5.8. Server-side Key Generation

Constrained devices sometimes do not have the necessary hardware to generate statistically random numbers for private keys and DTLS ephemeral keys. Past experience has also shown that low-resource endpoints sometimes generate numbers which could allow someone to decrypt the communication or guess the private key and impersonate as the device [PsQs] [RSAorig]. Additionally, random number key generation is costly, thus energy draining. Even though the random numbers that constitute the identity/cert do not get generated often, an endpoint may not want to spend time and energy generating keypairs, and just ask for one from the server.

In these scenarios, server-side key generation can be used. The client asks for the server or proxy to generate the private key and the certificate which are transferred back to the client in the server-side key generation response. In all respects, the server SHOULD treat the CSR as it would treat any enroll or re-enroll CSR; the only distinction here is that the server MUST ignore the public key values and signature in the CSR. These are included in the

request only to allow re-use of existing codebases for generating and parsing such requests.

The client /skg request needs to communicate to the server the Content-Format of the application/multipart-core elements. The key Content-Format requested by the client is depicted in the PKCS#10 request. If the request contains SMIMECapabilities the client is expecting Content-Format 280. Otherwise he expects a PKCS#8 key in Content-Format 284. The client expresses the preferred certificate Content-Format in his /skg request by using an Accept Option. The Accept Option is 281 when preferring a certificate in a PKCS#7 container or TBD287 when preferring a single X.509 certificate.

[RFC7030] provides two methods, symmetric and asymmetric, to optionally encrypt the generated key. The methods are signaled by the client by using the relevant attributes (SMIMECapabilities and DecryptKeyIdentifier or AsymmetricDecryptKeyIdentifier) in the CSR request. The symmetric key or the asymmetric keypair establishment method is out of scope of the specification.

The EST-coaps server-side key generation response is returned with Content-Format application/multipart-core [I-D.ietf-core-multipart-ct] containing a CBOR array with four items Section 5.3. The certificate part exactly matches the response from an enrollment response. The private key can be in unprotected PKCS#8 [RFC5958] format (Content-Format 284) or protected inside of CMS SignedData (Content-Format 280). The SignedData is signed by the party that generated the private key, which may be the EST server or the EST CA. The SignedData is further protected by placing it inside of a CMS EnvelopedData as explained in Section 4.4.2 of [RFC7030]. In summary, the symmetrically encrypted key is included in the encryptedKey attribute in a KEKRecipientInfo structure. In the case where the asymmetric encryption key is suitable for transport key operations the generated private key is encrypted with a symmetric key which is encrypted by the client defined (in the CSR) asymmetric public key and is carried in an encryptedKey attribute in a KeyTransRecipientInfo structure. Finally, if the asymmetric encryption key is suitable for key agreement, the generated private key is encrypted with a symmetric key which is encrypted by the client defined (in the CSR) asymmetric public key and is carried in an recipientEncryptedKeys attribute in a KeyAgreeRecipientInfo.

[RFC7030] recommends the use of additional encryption of the returned private key. For the context of this specification, clients and servers that choose to support server-side key generation MUST support unprotected (PKCS#8) private keys (Content-Format 284). Symmetric or asymmetric encryption of the private key (CMS EnvelopedData, Content-Format 280) SHOULD be supported for

deployments where end-to-end encryption needs to be provided between the client and a server. Such cases could include architectures where an entity between the client and the CA terminates the DTLS connection (Registrar in Figure 4).

6. DTLS Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that secures the exchanged CoAP messages. DTLS is one such secure protocol. EST depended in TLS. No other changes are necessary regarding the secure transport of EST messages.

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 several records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

The DTLS handshake is authenticated by using certificates. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in Section 3 of [RFC7030].

CoAP and DTLS can provide proof-of-identity for EST-coaps clients and servers with simple PKI messages as described in Section 3.1 of [RFC5272]. Moreover, 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 consists of the contents of the first "Finished" message in the (D)TLS handshake between server and client [RFC5929]. The client adds the "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request [RFC5967] to prove that the client is indeed in control of the private key at the time of the (D)TLS session establishment.

In the case of EST-coaps, the same operations can be performed during the DTLS handshake. For DTLS 1.2, in the event of handshake message fragmentation, the Hash of the handshake messages used in the MAC calculation of the Finished message MUST be computed as if each handshake message had been sent as a single fragment (Section 4.2.6 of [RFC6347]). The Finished message is calculated as shown in Section 7.4.9 of [RFC5246]. Similarly, for DTLS 1.3, the Finished message MUST be computed as if each handshake message had been sent as a single fragment (Section 5.8 of [I-D.ietf-tls-dtls13]) following the algorithm described in 4.4.4 of [RFC8446].

In a constrained CoAP environment, endpoints can't always 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. To alleviate this situation, an EST-coaps DTLS connection MAY remain open for sequential EST transactions. For example, an EST csrattrs request that is followed by a simpleenroll request can use the same authenticated DTLS connection. However, when a cacerts request is included in the set of sequential EST transactions, some additional security considerations apply regarding the use of the Implicit and Explicit TA database as explained in Section 11.1.

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. In some cases, like NAT rebinding, keeping the state of a connection is not possible when devices sleep for extended periods of time. In such occasions, [I-D.ietf-tls-dtls-connection-id] negotiates a connection ID that can eliminate the need for new handshake and its additional cost.

7. HTTPS-CoAPS Registrar

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 which case it will support TLS/HTTP instead of CoAPS. 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 Registrar at the edge is required to operate between the CoAP environment and the external HTTP network as shown in Figure 4.

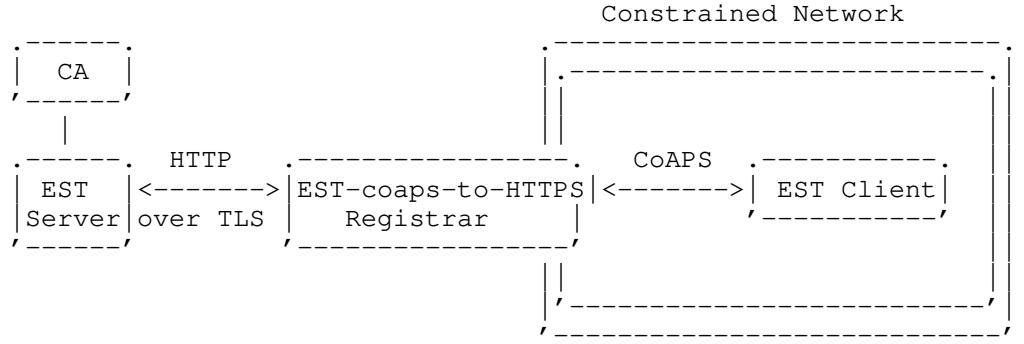


Figure 4: EST-coaps-to-HTTPS Registrar at the CoAP boundary.

The EST-coaps-to-HTTPS Registrar MUST terminate EST-coaps downstream and initiate EST connections over TLS upstream. The Registrar MUST authenticate and OPTIONALLY authorize the clients and it MUST be authenticated by the EST server or CA. The trust relationship between the Registrar and the EST server SHOULD be pre-established for the Registrar to proxy these connections on behalf of various clients.

When enforcing Proof-of-Possession (POP) linking, the DTLS tls-unique value of the (D)TLS session is used to prove that the private key corresponding to the public key is in the possession of and was used to establish the connection by the client as explained in Section 6). The POP linking information is lost between the EST-coaps client and the EST server when a Registrar is present. The EST server becomes aware of the presence of a Registrar from its TLS client certificate that includes id-kp-cmcRA [RFC6402] extended key usage extension (EKU). As explained in Section 3.7 of [RFC7030], the EST server SHOULD apply an authorization policy consistent with a Registrar client. For example, it could be configured to accept POP linking information that does not match the current TLS session because the authenticated EST client Registrar has verified this information when acting as an EST server.

For some use cases, clients that leverage server-side key generation might prefer for the enrolled keys to be generated by the Registrar if the CA does not support server-side key generation. In these cases, the Registrar MUST support random number generation using proper entropy. Such Registrar is responsible for generating a new CSR signed by a new key which will be returned to the client along with the certificate from the CA.

Table 1 contains the URI mappings between EST-coaps and EST that the Registrar MUST adhere to. Section 5.5 of this specification and Section 7 of [RFC8075] define the mappings between EST-coaps and HTTP response codes, that determine how the Registrar MUST translate CoAP response codes from/to HTTP status codes. The mapping from CoAP Content-Format to HTTP Media-Type is defined in Section 10.1. Additionally, a conversion from CBOR major type 2 to Base64 encoding MUST take place at the Registrar when server-side key generation is supported. If CMS end-to-end encryption is employed for the private key, the encrypted CMS EnvelopedData blob MUST be converted to binary in CBOR type 2 downstream to the client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP Registrar MUST reassemble the BLOCKs before translating the binary content to Base64, and consecutively relay the message upstream.

For the discovery of the EST server by the EST client in the CoAP environment, the EST-coaps-to-HTTP Registrar MUST announce itself according to the rules in Section 5.1. The available actions of the Registrars MUST be announced with as many resource paths necessary.

8. Parameters

This section addresses transmission parameters described in sections 4.7 and 4.8 of [RFC7252]. EST does not impose any unique values on the CoAP parameters in [RFC7252], but the EST parameter values need to be tuned to the CoAP parameter values.

It is RECOMMENDED, based on experiments, to follow the default CoAP configuration parameters ([RFC7252]). However, depending on the implementation scenario, retransmissions and timeouts can also occur on other networking layers, governed by other configuration parameters. A change in a server parameter MUST ensure the adjusted value is also available to all the endpoints with which these adjusted values are to be used to communicate.

Some further comments about some specific parameters, mainly from Table 2 in [RFC7252]:

- NSTART: Limit the number of simultaneous outstanding interactions that a client maintains to a given server. EST-coaps clients SHOULD use 1, which is the default. A EST-coaps client is not expected to interact with more than one servers at the same time.

- DEFAULT_LEISURE: This setting is only relevant in multicast scenarios, outside the scope of EST-coaps.

- PROBING_RATE: A parameter which specifies the rate of re-sending non-confirmable messages. The EST messages are defined to be sent as CoAP confirmable messages, hence this setting is not applicable.

Finally, the Table 3 parameters in [RFC7252] are mainly derived from Table 2. Directly changing parameters on one table would affect parameters on the other.

9. Deployment limitations

Although EST-coaps paves the way for the utilization of EST by constrained devices in constrained networks, some classes of devices [RFC7228] 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/DTLS/CoAP. It is

up to the network designer to decide which devices execute the EST protocol and which do not.

10. IANA Considerations

10.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry [COREparams] are specified in Table 3. These have been registered provisionally in the Expert Review range (0-255).

HTTP Media-Type	ID	Reference
application/pkcs7-mime; smime-type=server-generated-key	280	[RFC7030] [I-D.ietf-lamps-rfc5751-bis]
application/pkcs7-mime; smime-type=certs-only	281	[I-D.ietf-lamps-rfc5751-bis]
application/pkcs7-mime; smime-type=CMC-request	282	[RFC5273] [I-D.ietf-lamps-rfc5751-bis]
application/pkcs7-mime; smime-type=CMC-response	283	[RFC5273] [I-D.ietf-lamps-rfc5751-bis]
application/pkcs8	284	[RFC5958] [I-D.ietf-lamps-rfc5751-bis]
application/csrattrs	285	[RFC7030] [RFC7231]
application/pkcs10	286	[RFC5967] [I-D.ietf-lamps-rfc5751-bis]
application/pkix-cert	TBD287	[RFC2585]

Table 3: Table 3: New CoAP Content-Formats

The Content-Formats 281 to 286 have been the subject of an earlier temporary allocation. It is suggested that 287 is allocated to TBD287.

10.2. Resource Type registry

This memo registers a new Resource Type (rt=) Link Target Attributes in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" registry.

- rt="ace.est". This EST resource is used to query and return the supported EST resources of a CoAP server.

- rt="ace.est.crts". This resource depicts the support of EST get cacerts.
- rt="ace.est.sen". This resource depicts the support of EST simple enroll.
- rt="ace.est.sren". This resource depicts the support of EST simple reenroll.
- rt="ace.est.att". This resource depicts the support of EST CSR attributes.
- rt="ace.est.skg". This resource depicts the support of EST server-side key generation.

11. Security Considerations

11.1. 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. The transport of these keys is inherently risky. Analysis SHOULD be done to establish whether server-side key generation enhances or decreases the probability of identity stealing.

It is also RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication is carefully managed to reduce the chance of a third-party CA with poor certification practices jeopardizing authentication. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3 of [RFC7030]) limits any risk to the first DTLS exchange. Alternatively, in a case where a /sen request immediately follows a /crt, a client MAY choose to keep the connection authenticated by the Implicit TA open for efficiency reasons (Section 6). A client that pipelines EST-coaps /crt request with other requests in the same DTLS connection SHOULD revalidate the server certificate chain against the updated Explicit TA from the /crt response before proceeding with the subsequent requests. If the server certificate chain does not authenticate against the database, the client SHOULD close the connection without completing the rest of the requests. The updated Explicit TA MUST continue to be used in new DTLS connections.

In cases where the IDevID used to authenticate the client is expired the server MAY still authenticate the client because IDevIDs are expected to live as long as the device itself (Section 4). In such occasions, checking the certificate revocation status or authorizing the client using another method is important for the server to ensure that the client is to be trusted.

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 certificate request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that only the authenticated client currently has access to the private key.

What's more, POP linking uses tls-unique as it is defined in [RFC5929]. The 3SHAKE attack [tripleshake] poses a risk by allowing a man-in-the-middle to leverage session resumption and renegotiation to inject himself between a client and server even when channel binding is in use. The attack was possible because of certain (D)TLS implementation imperfections. In the context of this specification, an attacker could invalidate the purpose of the POP linking ChallengePassword in the client request by resuming an EST-coaps connection. Even though the practical risk of such an attack to EST-coaps is not devastating, we would rather use a more secure channel binding mechanism. Such a mechanism could include an updated tls-unique value generation like the tls-unique-prf defined in [I-D.josefsson-sasl-tls-cb] by using a TLS exporter [RFC5705] in TLS 1.2 or TLS 1.3's updated exporter (Section 7.5 of [RFC8446]). Such mechanism has not been standardized yet. Adopting in this document a channel binding value generated from an exporter would break backwards compatibility. Thus, in this specification we still depend in the tls-unique mechanism defined in [RFC5929], especially since the practicality of such an attack would not expose any messages exchanged with EST-coaps.

Regarding the Certificate Signing Request (CSR), a 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.

11.2. HTTPS-CoAPS Registrar considerations

The Registrar proposed in Section 7 must be deployed with care, and only when the recommended connections are impossible. When POP linking is used the Registrar terminating the TLS connection establishes a new one with the upstream CA. Thus, it is impossible for POP linking to be enforced end-to-end for the EST transaction. The EST server could be configured to accept POP linking information that does not match the current TLS session because the authenticated EST Registrar client has verified this information when acting as an EST server.

The introduction of an EST-coaps-to-HTTP Registrar assumes the client can trust the registrar using its implicit or explicit TA database. It also assumes the Registrar has a trust relationship with the upstream EST server in order to act on behalf of the clients. When a client uses the Implicit TA database for certificate validation, he SHOULD confirm if the server is acting as an RA by the presence of the id-kp-cmcRA [RFC6402] EKU in the server certificate. If the server certificate does not include the EKU, it is RECOMMENDED that the client includes "Linking Identity and POP Information" (Section 6) in requests.

In a server-side key generation case, if no end-to-end encryption is used, the Registrar may be able see the private key as it acts as a man-in-the-middle. Thus, the client puts its trust on the Registrar not exposing the private key.

Clients that leverage server-side key generation without end-to-end encryption of the private key (Section 5.8) have no knowledge if the Registrar will be generating the private key and enrolling the certificates with the CA or if the CA will be responsible for generating the key. In such cases, the existence of a Registrar requires the client to put its trust on the registrar doing the right thing if it is generating the private key.

12. Contributors

Martin Furuhed contributed to the EST-coaps specification by providing feedback based on the Nexus EST over CoAPS server implementation that started in 2015. Sandeep Kumar kick-started this specification and was instrumental in drawing attention to the importance of the subject.

13. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS and his support for the Content-Format specification. 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 for his feedback on technical details of the solution. Constructive comments were received from Benjamin Kaduk, Eliot Lear, Jim Schaad, Hannes Tschofenig, Julien Vermillard, John Manuel, Oliver Pfaff and Pete Beal.

Interop tests were done by Oliver Pfaff, Thomas Werner, Oskar Camezind, Bjorn Elmers and Joel Hoglund.

Robert Moskowitz provided code to create the examples.

14. References

14.1. Normative References

[I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-02 (work in progress), August 2018.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-30 (work in progress), November 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>.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, DOI 10.17487/RFC2585, May 1999, <https://www.rfc-editor.org/info/rfc2585>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

14.2. Informative References

[COREparams]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml>.

[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-19 (work in progress), January 2019.

[I-D.ietf-lamps-rfc5751-bis]
Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", draft-ietf-lamps-rfc5751-bis-12 (work in progress), September 2018.

[I-D.ietf-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-02 (work in progress), October 2018.

[I-D.josefsson-sasl-tls-cb]
Josefsson, S., "Channel Bindings for TLS based on the PRF", draft-josefsson-sasl-tls-cb-03 (work in progress), March 2015.

[I-D.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", draft-moskowitz-ecdsa-pki-04 (work in progress), September 2018.

[ieee802.15.4]
Institute of Electrical and Electronics Engineers, "IEEE Standard 802.15.4-2006", 2006.

[ieee802.1ar]
Institute of Electrical and Electronics Engineers, "IEEE 802.1AR Secure Device Identifier", December 2009.

[PsQs] Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", USENIX Security Symposium 2012 ISBN 978-931971-95-9, August 2012.

[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>.

[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>.

[RFC5273] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC): Transport Protocols", RFC 5273, DOI 10.17487/RFC5273, June 2008, <https://www.rfc-editor.org/info/rfc5273>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[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>.

[RFC6402] Schaad, J., "Certificate Management over CMS (CMC) Updates", RFC 6402, DOI 10.17487/RFC6402, November 2011, <https://www.rfc-editor.org/info/rfc6402>.

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[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>.

[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>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RSAorig] Petr Svenda, Matus Nemec, Peter Sekan, Rudolf Kvasnovsky, David Formanek, David Komarek, Vashek Matyas, "The Million-Key Question - Investigating the Origins of RSA Public Keys", USENIX Security Symposium 2016 ISBN 978-1-931971-32-4, August 2016.

[tripleshake] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cedric Fournet, Alfredo Pironti, Pierre-Yves Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS", IEEE Security and Privacy ISBN 978-1-4799-4686-0, May 2014.

Appendix A. EST messages to EST-coaps

This section shows similar examples to the ones presented in Appendix A of [RFC7030]. The payloads in the examples are the hex encoded binary, generated with 'xxd -p', of the PKI certificates created following [I-D.moskowitz-ecdsa-pki]. The payloads are shown unencrypted. In practice the message content would be binary

formatted and transferred over an encrypted DTLS tunnel. The hexadecimal representations in the examples below would NOT be transported in hex, but in binary. Hex is used for visualization purposes because a binary representation cannot be rendered well in text.

The certificate responses included in the examples contain Content-Format 281 (application/pkcs7). If the client had requested Content-Format TBD287 (application/pkix-cert) with an Accept Option, the server would respond a single DER binary certificate.

These examples assume that the resource discovery, returned a short base path of "/est".

The corresponding CoAP headers are only shown in Appendix A.1. Creating CoAP headers is assumed to be generally understood.

The message content breakdown is presented in Appendix C.

A.1. cacerts

In EST-coaps, a cacerts message can be:

GET coaps://est-coaps.example.ietf.org:9085/est/crts
(Accept: 281)

The corresponding CoAP header fields are shown below. The use of block and DTLS are worked out in Appendix B.

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Token = 0x9a (client generated)
Options
Option (Uri-Host)               [optional]
   Option Delta = 0x3  (option# 3)
   Option Length = 0x9
   Option Value = est-coaps.example.ietf.org
Option (Uri-Port)               [optional]
   Option Delta = 0x4  (option# 3+4=7)
   Option Length = 0x4
   Option Value = 9085
 Option (Uri-Path)
   Option Delta = 0x4   (option# 7+4=11)
   Option Length = 0x5
   Option Value = "est"
 Option (Uri-Path)
   Option Delta = 0x0   (option# 11+0=11)
   Option Length = 0x6
   Option Value = "crts"
 Option (Accept)
   Option Delta = 0x6   (option# 11+6=17)
   Option Length = 0x2
   Option Value = 281
Payload = [Empty]
```

The Uri-Host and Uri-Port Options are optional. They are usually omitted as the DTLS destination and port are sufficient. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

A 2.05 Content response with a cert in EST-coaps will then be

```
2.05 Content (Content-Format: 281)
   {payload with certificate in binary format}
```

with CoAP fields

```
Ver = 1
T = 2 (ACK)
Code = 0x45 (2.05 Content)
Token = 0x9a   (copied from request by server)
Options
  Option (Content-Format)
    Option Delta = 0xC  (option# 12)
    Option Length = 0x2
    Option Value = 281

[ The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text. ]

  Payload =
3082027b06092a864886f70d010702a082026c308202680201013100300b
06092a864886f70d010701a082024e3082024a308201f0a0030201020209
009189bcdf9c99244b300a06082a8648ce3d0403023067310b3009060355
040613025553310b300906035504080c024341310b300906035504070c02
4c413114301206035504a0c0b4578616d706c6520496e63311630140603
55040b0c0d6365727469666963617469f6e3110300e06035504030c0752
6f6f74204341301e170d3139303130373130343034315a170d3339303130
323130343034315a3067310b300906035504061302555331 0b3009060355
04080c024341310b300906035504070c024c413114301206035504a0c0b
4578616d706c6520496e633116301406035504b0c0d6365727469666963
6174696f6e3110300e06035504030c07526f6f74204341305930130607 2a
8648ce3d020106082a8648ce3d030107034200048149940 82b6e8185f3df
53f5e0bee698973335200023ddf78cd17a443ffd8ddd40908769c55652ac
2ccb75c4a50a7c7ddb7c22dae6c85cca538209fdbbf104c9a38184308181
301d0603551d0e041604142495e816ef6ffcaaf356ce4adffe33cf492abb
a8301f0603551d2304183016801424 95e816ef6ffcaaf356ce4adffe33cf
492abba8300f0603551d130101ff040530030101ff300e0603551d0f0101
ff040403020106301e0603551d1104173015811363657274696679406578
616d706c652e636f6d300a06082a8648ce3d040302034800304502210 0da
e37c96f154c32ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f135327
2f022047a28ae5c7306163b3c3834bab3c103f743070594c089aaa0ac870
cd13b902caa1003100
```

The breakdown of the payload is shown in Appendix C.1.

A.2. enroll / reenroll

During the (re-)enroll exchange the EST-coaps client uses a CSR (Content-Format 286) request in the POST request payload. The Accept option tells the server that the client is expecting Content-Format 281 (PKCS#7) in the response. As shown in Appendix C.2, the CSR contains a ChallengePassword which is used for POP linking (Section 6).

```
POST [2001:db8::2:1]:61616/est/sen
(Token: 0x45)
(Accept: 281)
(Content-Format: 286)

[ The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text. ]

3082018b30820131020100305c310b300906035504061302555331 0b3009
06035504080c024341310b300906035504070c024c413114301206035504
0a0c0b6578616d706c6520496e63310c300a060355040b0c03496f54310f
300d06035504051306577431323334305930130607 2a8648ce3d02010608
2a8648ce3d03010703420004c8b421f11c25e47e3ac57123bf2d9fdc494f
028bc351cc80c03f150bf50cff958d75419d81a6a245dffae790be95cf75
f602f9152618f816a2b23b5638e59fd9a073303406092a864886f70d0109
0731270c2576437630292a264a4b4a3bc3a2c280c2992f3e3c2e2c3d6b6e
7634332323403d204e787e60303b06092a864886f70d01090e312e302c30
2a0603551d1104233021a01f06082b06010505070804a013301106092b06
010401b43b0a0104040102030430 0a06082a8648ce3d0403020348003045
02210092563a546463bd9ecff170d0fd1f2ef0d3d012160e5ee90cffedab
ec9b9a38920220179f10a3436109051abad17590a09bc87c4dce5453a6fc
1135a1e84eed754377
```

After verification of the CSR by the server, a 2.01 Content response with the issued certificate will be returned to the client.

```
2.01 Created
(Token: 0x45)
(Content-Format: 281)

[ The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text. ]

3082026e06092a864886f70d010702a082025f3082025b0201013100300b
06092a864886f70d010701a08202413082023d308201e2a0030201020208
7e7661d7b54e4632300a06082a8648ce3d040302305d310b300906035504
0613025553310b300906035504080c024341311143012060355040a0c0b45
78616d706c6520496e6331163014060355040b0c0d636572746966696361
74696f6e3113301106035504030c0a3830322e3141522043413020170d31
393031333131313231365a180f393939393132333132333353935395a30
5c310b300906035504061302555310b300906035504080c024341310b30
0906035504070c024c4131143012060355040a0c0b6578616d706c652049
6e63310c300a060355040b0c03496f54310f300d06035504051306577431
323334305930130607 2a8648ce3d020106082a8648ce3d03010703420004
c8b421f11c25e47e3ac57123bf2d9fdc494f028bc351cc80c03f150bf50c
ff958d75419d81a6a245dffae790be95cf75f602f9152618f816a2b23b56
38e59fd9a3818a30818730090603551d1304023000301d0603551d0e0416
041496600d8716bf7fd0e752d0ac760777ad665d02a0301f0603551d2304
183016801468d16551f951bfc82a431d0d9f08bc2d205b1160300e060355
1d0f0101ff0404030205a0302a0603551d1104233021a01f06082b060105
05070804a013301106092b06010401b43b0a01040401020304300a06082a
8648ce3d040302034900303046022100c0d81996d2507d693f3c48eaa5ee94
91bda6db214099d98117c63b361374cd86022100a774989f4c321a5cf25d
832a4d336a08ad67df20f1506421188a0ade6d349236a1003100
```

The breakdown of the request and response is shown in Appendix C.2.

As described in Section 5.7, if the server is not able to provide a response immediately, it sends an empty ACK with response code 5.03 (Service Unavailable) and the Max-Age Option. See Figure 3 for an example exchange.

A.3. serverkeygen

In a serverkeygen exchange the CoAP POST request looks like

```
POST coaps://192.0.2.1:8085/est/skg
(Token: 0xa5)
(Accept: 281)
(Content-Format: 286)

[ The hexadecimal representation below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text. ]

3081cf30780201003016311430120603550 40a0c0b736b67206578616d70
6c653059301306072a8648ce3d020106082a8648ce3d030107034200041b
b8c1117896f98e4506c03d70efbe820d8e38ea97e9d65d52c8460c5852c5
1dd89a61370a2843760fc859799d78cd33f3c1846e304f1717f8123f1a28
4cc99fa000300a06082a8648ce3d040302034700304402203 87cd4e9cf62
8d4af77f92ebed4890d9d141dca86cd2757dd14cbd59cdf6961802202f24
5e828c77754378b66660a4977f113cacdaa0cc7bad7d1474a7fd155d090d
```

The response would follow [I-D.ietf-core-multipart-ct] and could look like

```
2.01 Content
(Token: 0xa5)
(Content-Format: 62)

[ The hexadecimal representations below would NOT be transported
in hex, but in binary. Hex is used because a binary representation
cannot be rendered well in text. ]

84                                  # array(4)
19 011C                             # unsigned(284)
58 8A                               # bytes(138)
308187020100301306072a8648ce3d020106082a8648ce3d030107046d30
6b02010104200b9a67785b65e07360b6d28cfc1d3f3925c0755799deeca7
45372b01697bd8a6a144034200041bb8c1117896f98e4506c03d70efbe82
0d8e38ea97e9d65d52c8460c5852c51dd89a61370a2843760fc859799d78
cd33f3c1846e304f1717f8123f1a284cc99f
19 0119                             # unsigned(281)
59 01D3                             # bytes(467)
308201cf06092a864886f70d010702a08201c0308201bc0201013100300b
06092a864886f70d010701a08201a23082019e30820143a0030201020208
126de8571518524b300a06082a8648ce3d040302301631143012060355040
0a0c0b736b67206578616d706c65301e170d313930313039303835373038
5a170d333930313034303853730385a3016311430120603550400a0c0b73
6b67206578616d706c653059301306072a8648ce3d020106082a8648ce3d
030107034200041bb8c1117896f98e4506c03d70efbe820d8e38ea97e9d6
5d52c8460c5852c51dd89a61370a2843760fc859799d78cd33f3c1846e30
4f1717f8123f1a284cc99fa37b307930090603551d1304023000302c0609
6086480186f842010d041f161d4f70656e53534c2047656e657261746564
204365727469666963617465301d0603551d0e04160414494be598dc8dbc
0dbc071c486b777460e5cce621301f0603551d23041830168014494be598
dc8dbc0dbc071c486b777460e5cce621300a06082a8648ce3d0403020349
003046022100a4b167d0f9add9202810e6bf6a290b8cfdfc9b9c9fea2cc1
c8fc3a464f79f2c202210081d31ba142751a7b4a34fd1a01fcfb08716b9e
b53bdaadc9ae60b08f52429c0fa1003100
```

The private key in the response above is without CMS EnvelopedData and has no additional encryption beyond DTLS (Section 5.8).

The breakdown of the request and response is shown in Appendix C.3

A.4. csrattrs

Below is a csrattrs exchange

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att

RES:
2.05 Content
(Content-Format: 285)
```

[The hexadecimal representation below would NOT be transported in hex, but in binary. Hex is used because a binary representation cannot be rendered well in text.]

```
307c06072b06010101011630220603883701311b131950617273652053455
420617320322e3939392e3120646174610609 2a864886f70d010907302c06
038837023125060388370306038837041319506172736520534554206173 2
0322e3939392e3220646174610609 2b240303020801010b06096086480165
03040202
```

A 2.05 Content response should contain attributes which are relevant for the authenticated client. This example is copied from section A.2 in [RFC7030], where the base64 representation is replaced with a hexadecimal representation of the equivalent binary format. The EST-coaps server returns attributes that the client can ignore if they are unknown to him.

Appendix B. EST-coaps Block message examples

Two examples are presented in this section:

1. a cacerts exchange shows the use of Block2 and the block headers
2. an enroll exchange shows the Block1 and Block2 size negotiation for request and response payloads.

The payloads are shown unencrypted. In practice the message contents would be binary formatted and transferred over an encrypted DTLS tunnel. The corresponding CoAP headers are only shown in Appendix B.1. Creating CoAP headers are assumed to be generally known.

B.1. cacerts

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 cacerts exchange over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030]

contains 639 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 9 packets with a payload of 64 bytes each, followed by a last tenth packet of 63 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP request 10 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 (block-option:NUM/M/size) indicating the kind of Block Option (2 in this case) followed by a colon, and then the block number (NUM), the more bit (M = 0 in Block2 response means it is last block), and block size with exponent (2**(SZX+4)) separated by slashes. The Length 64 is used with SZX=2 to avoid IP fragmentation. The CoAP Request is sent confirmable (CON) and the Content-Format of the response, even though not shown, is 281 (application/pkcs7-mime; smime-type=certs-only). The transfer of the 10 blocks with partially filled block NUM=9 is shown below

```
 GET coaps://est-coaps.example.ietf.org:9085/est/crts (2:0/0/64)  -->
                <--   (2:0/1/64) 2.05 Content
 GET coaps://est-coaps.example.ietf.org:9085/est/crts (2:1/0/64)  -->
                <--   (2:1/1/64) 2.05 Content
                              |
                              |
                              |
 GET coaps://est-coaps.example.ietf.org:9085/est/crts (2:9/0/64) -->
                <--   (2:9/0/64) 2.05 Content
```

The header of the GET request looks like

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.1 GET)
Token = 0x9a    (client generated)
Options
 Option (Uri-Host)            [optional]
   Option Delta = 0x3  (option# 3)
   Option Length = 0x9
   Option Value = est-coaps.example.ietf.org
 Option (Uri-Port)            [optional]
   Option Delta = 0x4   (option# 3+4=7)
   Option Length = 0x4
   Option Value = 9085
 Option (Uri-Path)
   Option Delta = 0x4    (option# 7+4=11)
   Option Length = 0x5
   Option Value = "est"
 Option (Uri-Path)Uri-Path)
   Option Delta = 0x0    (option# 11+0=11)
   Option Length = 0x6
   Option Value = "crts"
 Option (Accept)
   Option Delta = 0x6   (option# 11+6=17)
   Option Length = 0x2
   Option Value = 281
Payload = [Empty]
```

The Uri-Host and Uri-Port Options are optional. They are usually omitted as the DTLS destination and port are sufficient. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers and uses the Options to route the requests accordingly. Alternatively, if a UDP port to a server is blocked, someone could send the DTLS packets to a known open port on the server and use the Uri-Port to convey the intended port he is attempting to reach.

For further detailing the CoAP headers, the first two and the last blocks are written out below. The header of the first Block2 response looks like

```
    Ver = 1
    T = 2 (ACK)
    Code = 0x45 (2.05 Content)
    Token = 0x9a     (copied from request by server)
    Options
      Option
        Option Delta = 0xC  (option# 12 Content-Format)
        Option Length = 0x2
        Option Value = 281
      Option
        Option Delta = 0xB  (option# 12+11=23 Block2)
        Option Length = 0x1
        Option Value = 0x0A (block#=0, M=1, SZX=2)

    [ The hexadecimal representation below would NOT be transported
    in hex, but in binary. Hex is used because a binary representation
    cannot be rendered well in text. ]

    Payload =
  3082027b06092a864886f70d010702a082026c308202680201013100300b
  06092a864886f70d010701a082024e3082024a308201f0a0030201020209
  009189bc
```

The second Block2:

```
    Ver = 1
    T = 2 (means ACK)
    Code = 0x45 (2.05 Content)
    Token = 0x9a     (copied from request by server)
    Options
      Option
        Option Delta = 0xC  (option# 12 Content-Format)
        Option Length = 0x2
        Option Value = 281
      Option
        Option Delta = 0xB  (option 12+11=23 Block2)
        Option Length = 0x1
        Option Value = 0x1A (block#=1, M=1, SZX=2)

    [ The hexadecimal representation below would NOT be transported
    in hex, but in binary. Hex is used because a binary representation
    cannot be rendered well in text. ]

    Payload =
  df9c99244b300a06082a8648ce3d0403023067310b300906035504061302
  5553310b300906035504080c024341310b300906035504070c024c413114
  30120603
```

The 10th and final Block2:

```
  Ver = 1
  T = 2 (means ACK)
  Code = 0x45      (2.05 Content)
  Token = 0x9a     (copied from request by server)
  Options
    Option
      Option Delta = 0xC  (option# 12 Content-Format)
      Option Length = 0x2
      Option Value = 281
    Option
      Option Delta = 0xB  (option# 12+11=23 Block2 )
      Option Length = 0x2
      Option Value = 0x92 (block#=9, M=0, SZX=2)

  [ The hexadecimal representation below would NOT be transported
  in hex, but in binary. Hex is used because a binary representation
  cannot be rendered well in text. ]

  Payload =
2ec0b4af52d46f3b7ecc9687ddf267bcec368f7b7f1353272f022047a28a
e5c7306163b3c3834bab3c103f743070594c089aaa0ac870cd13b902caa1
003100
```

B.2. enroll / reenroll

In this example the requested Block2 size of 256 bytes, required by the client, is transferred to the server in the very first request message. The block size 256=(2**(SZX+4)) which gives SZX=4. The notation for block numbering is the same as in Appendix B.1. It is assumed that CSR takes N1+1 blocks and the cert response takes N2+1 blocks. The header fields and the payload are omitted for brevity.

```
   POST [2001:db8::2:1]:61616/est/sen (CON)(1:0/1/256) {CSR req} -->
          <-- (ACK) (1:0/1/256) (2.31 Continue)
   POST [2001:db8::2:1]:61616/est/sen (CON)(1:1/1/256) {CSR req} -->
          <-- (ACK) (1:1/1/256) (2.31 Continue)
                           .
                           .
                           .
   POST [2001:db8::2:1]:61616/est/sen (CON)(1:N1/0/256){CSR req} -->
          <-- (ACK) (1:N1/0/256)(2:0/1/256)(2.04 Changed){Cert resp}
   POST [2001:db8::2:1]:61616/est/sen (CON)(2:1/0/256)           -->
          <-- (ACK) (2:1/1/256)(2.04 Changed) {Cert resp}
                           .
                           .
                           .
   POST [2001:db8::2:1]:61616/est/sen (CON)(2:N2/0/256)          -->
          <-- (ACK) (2:N2/0/256) (2.04 Changed) {Cert resp}
```

Figure 5: EST-COAP enrollment with multiple blocks

N1+1 blocks have been transferred from client to the server and N2+1 blocks have been transferred from server to client.

Appendix C. Message content breakdown

This appendix presents the breakdown of the hexadecimal dumps of the binary payloads shown in Appendix A.

C.1. cacerts

Breakdown of cacerts response containing one root CA certificate.

```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number:
               91:89:bc:df:9c:99:24:4b
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: C=US, ST=CA, L=LA, O=Example Inc,
                   OU=certification, CN=Root CA
           Validity
               Not Before: Jan  7 10:40:41 2019 GMT
               Not After : Jan  2 10:40:41 2039 GMT
           Subject: C=US, ST=CA, L=LA, O=Example Inc,
                    OU=certification, CN=Root CA
           Subject Public Key Info:
               Public Key Algorithm: id-ecPublicKey
                   Public-Key: (256 bit)
                   pub:
                       04:81:49:94:08:2b:6e:81:85:f3:df:53:f5:e0:be:
                       e6:98:97:33:35:20:00:23:dd:f7:8c:d1:7a:44:3f:
                       fd:8d:dd:40:90:87:69:c5:56:52:ac:2c:cb:75:c4:
                       a5:0a:7c:7d:db:7c:22:da:e6:c8:5c:ca:53:82:09:
                       fd:bb:f1:04:c9
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           X509v3 extensions:
               X509v3 Subject Key Identifier:
   24:95:E8:16:EF:6F:FC:AA:F3:56:CE:4A:DF:FE:33:CF:49:2A:BB:A8
               X509v3 Authority Key Identifier:
                   keyid:
   24:95:E8:16:EF:6F:FC:AA:F3:56:CE:4A:DF:FE:33:CF:49:2A:BB:A8

               X509v3 Basic Constraints: critical
                   CA:TRUE
               X509v3 Key Usage: critical
                   Certificate Sign, CRL Sign
               X509v3 Subject Alternative Name:
                   email:certify@example.com
       Signature Algorithm: ecdsa-with-SHA256
            30:45:02:21:00:da:e3:7c:96:f1:54:c3:2e:c0:b4:af:52:d4:
            6f:3b:7e:cc:96:87:dd:f2:67:bc:ec:36:8f:7b:7f:13:53:27:
            2f:02:20:47:a2:8a:e5:c7:30:61:63:b3:c3:83:4b:ab:3c:10:
            3f:74:30:70:59:4c:08:9a:aa:0a:c8:70:cd:13:b9:02:ca
```

C.2. enroll / reenroll

The breakdown of the request is

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=CA, L=LA, O=example Inc,
                                OU=IoT/serialNumber=Wt1234
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
                    9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
                    0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
                    be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
                    56:38:e5:9f:d9
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            challengePassword        : <256-bit POP linking value>
        Requested Extensions:
            X509v3 Subject Alternative Name:
                othername:<unsupported>
    Signature Algorithm: ecdsa-with-SHA256
         30:45:02:21:00:92:56:3a:54:64:63:bd:9e:cf:f1:70:d0:fd:
         1f:2e:f0:d3:d0:12:16:0e:5e:e9:0c:ff:ed:ab:ec:9b:9a:38:
         92:02:20:17:9f:10:a3:43:61:09:05:1a:ba:d1:75:90:a0:9b:
         c8:7c:4d:ce:54:53:a6:fc:11:35:a1:e8:4e:ed:75:43:77
```

The CSR contained a ChallengePassword which is used for POP linking (Section 6).

The breakdown of the issued certificate response is

```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number: 9112578475118446130 (0x7e7661d7b54e4632)
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: C=US, ST=CA, O=Example Inc, OU=certification,
                                CN=802.1AR CA
           Validity
               Not Before: Jan 31 11:29:16 2019 GMT
               Not After : Dec 31 23:59:59 9999 GMT
           Subject: C=US, ST=CA, L=LA, O=example Inc,
                                 OU=IoT/serialNumber=Wt1234
           Subject Public Key Info:
               Public Key Algorithm: id-ecPublicKey
                   Public-Key: (256 bit)
                   pub:
                       04:c8:b4:21:f1:1c:25:e4:7e:3a:c5:71:23:bf:2d:
                       9f:dc:49:4f:02:8b:c3:51:cc:80:c0:3f:15:0b:f5:
                       0c:ff:95:8d:75:41:9d:81:a6:a2:45:df:fa:e7:90:
                       be:95:cf:75:f6:02:f9:15:26:18:f8:16:a2:b2:3b:
                       56:38:e5:9f:d9
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           X509v3 extensions:
               X509v3 Basic Constraints:
                   CA:FALSE
               X509v3 Subject Key Identifier:
   96:60:0D:87:16:BF:7F:D0:E7:52:D0:AC:76:07:77:AD:66:5D:02:A0

               X509v3 Authority Key Identifier:
                   keyid:
   68:D1:65:51:F9:51:BF:C8:2A:43:1D:0D:9F:08:BC:2D:20:5B:11:60

               X509v3 Key Usage: critical
                   Digital Signature, Key Encipherment
               X509v3 Subject Alternative Name:
                   othername:<unsupported>
       Signature Algorithm: ecdsa-with-SHA256
            30:46:02:21:00:c0:d8:19:96:d2:50:7d:69:3f:3c:48:ea:a5:
            ee:94:91:bd:a6:db:21:40:99:d9:81:17:c6:3b:36:13:74:cd:
            86:02:21:00:a7:74:98:9f:4c:32:1a:5c:f2:5d:83:2a:4d:33:
            6a:08:ad:67:df:20:f1:50:64:21:18:8a:0a:de:6d:34:92:36
```

C.3. serverkeygen

The following is the breakdown of the request example used.

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: O=skg example
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
                    be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
                    52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                    9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                    1a:28:4c:c9:9f
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        Attributes:
            a0:00
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:38:7c:d4:e9:cf:62:8d:4a:f7:7f:92:eb:ed:48:
         90:d9:d1:41:dc:a8:6c:d2:75:7d:d1:4c:bd:59:cd:f6:96:18:
         02:20:2f:24:5e:82:8c:77:75:43:78:b6:66:60:a4:97:7f:11:
         3c:ac:da:a0:cc:7b:ad:7d:14:74:a7:fd:15:5d:09:0d
```

The following is the breakdown of the private key content of the server-side key generation response payload.

```
Private-Key: (256 bit)
priv:
    0b:9a:67:78:5b:65:e0:73:60:b6:d2:8c:fc:1d:3f:
    39:25:c0:75:57:99:de:ec:a7:45:37:2b:01:69:7b:
    d8:a6
pub:
    04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
    be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
    52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
    9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
    1a:28:4c:c9:9f
ASN1 OID: prime256v1
NIST CURVE: P-256
```

The following is the breakdown of the certificate of the second part of the server-side key generation response payload.

```
   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number: 1327972925857878603 (0x126de8571518524b)
       Signature Algorithm: ecdsa-with-SHA256
           Issuer: O=skg example
           Validity
               Not Before: Jan  9 08:57:08 2019 GMT
               Not After : Jan  4 08:57:08 2039 GMT
           Subject: O=skg example
           Subject Public Key Info:
               Public Key Algorithm: id-ecPublicKey
                   Public-Key: (256 bit)
                   pub:
                       04:1b:b8:c1:11:78:96:f9:8e:45:06:c0:3d:70:ef:
                       be:82:0d:8e:38:ea:97:e9:d6:5d:52:c8:46:0c:58:
                       52:c5:1d:d8:9a:61:37:0a:28:43:76:0f:c8:59:79:
                       9d:78:cd:33:f3:c1:84:6e:30:4f:17:17:f8:12:3f:
                       1a:28:4c:c9:9f
                   ASN1 OID: prime256v1
                   NIST CURVE: P-256
           X509v3 extensions:
               X509v3 Basic Constraints:
                   CA:FALSE
               Netscape Comment:
                   OpenSSL Generated Certificate
               X509v3 Subject Key Identifier:
   49:4B:E5:98:DC:8D:BC:0D:BC:07:1C:48:6B:77:74:60:E5:CC:E6:21
               X509v3 Authority Key Identifier:
                   keyid:
   49:4B:E5:98:DC:8D:BC:0D:BC:07:1C:48:6B:77:74:60:E5:CC:E6:21

       Signature Algorithm: ecdsa-with-SHA256
            30:46:02:21:00:a4:b1:67:d0:f9:ad:d9:20:28:10:e6:bf:6a:
            29:0b:8c:fd:fc:9b:9c:9f:ea:2c:c1:c8:fc:3a:46:4f:79:f2:
            c2:02:21:00:81:d3:1b:a1:42:75:1a:7b:4a:34:fd:1a:01:fc:
            fb:08:71:6b:9e:b5:3b:da:ad:c9:ae:60:b0:8f:52:42:9c:0f
```

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
URI: http://www.sandelman.ca/

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

ACE
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2019

M. Jones
Microsoft
L. Seitz
RISE SICS
G. Selander
Ericsson AB
S. Erdtman
Spotify
H. Tschofenig
ARM Ltd.

February 21, 2019

Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)
draft-ietf-ace-cwt-proof-of-possession-06

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

Table of Contents

1. Introduction

This specification describes how a CBOR Web Token (CWT) [RFC8392] can declare that the presenter of the CWT possesses a particular proof-of-possession (PoP) key. Proof of possession of a key is also sometimes described as being the holder-of-key. This specification provides equivalent functionality to "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)" [RFC7800], but using CBOR [RFC7049] and CWTs [RFC8392] rather than JSON [RFC8259] and JWTs [JWT].

2. Terminology

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

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

These terms are defined by this specification:

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

Presenter
: Party that proves possession of a private key (for asymmetric key cryptography) or secret key (for symmetric key cryptography) to a recipient.
In context of OAuth this party is also called OAuth Client.

Recipient
: Party that receives the CWT containing the proof-of-possession key information from the presenter.
In context of OAuth this party is also called OAuth Resource Server.

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

3.1. Confirmation Claim

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

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

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

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

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

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

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

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

```
  {
  /iss/ 1 : "coaps://server.example.com",
  /aud/ 3 : "coaps://client.example.org",
  /exp/ 4 : 1361398824,
  /cnf/ 8 :{
    /COSE_Key/ 1 :{
      /kty/ 1 : /EC/ 2,
      /crv/ -1 : /P-256/ 1,
      /x/ -2 : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa27c9
                 e354089bbe13',
      /y/ -3 : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020731e
                 79a3b4e47120'
     }
   }
 }
```

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

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

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

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

The following example (using CBOR diagnostic notation, with linebreaks for readability) illustrates a symmetric key that could subsequently be encrypted for use in the "Encrypted_COSE_Key" member:

```
    {
     /kty/ 1 : /Symmetric/ 4,
     /alg/ 3 : /HMAC256/ 5,
     /k/ -1 : h'6684523ab17337f173500e5728c628547cb37df
                e68449c65f885d1b73b49eae1'
    }
```

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

The following example CWT Claims Set of a CWT (using CBOR diagnostic notation, with linebreaks for readability) illustrates the use of an encrypted symmetric key as the "Encrypted_COSE_Key" member value:

```
{
 /iss/ 1 : "coaps://server.example.com",
 /sub/ 2 : "24400320",
 /aud/ 3: "s6BhdRkqt3",
 /exp/ 4 : 1311281970,
 /iat/ 5 : 1311280970,
 /cnf/ 8 : {
 /COSE_Encrypt0/ 2 : [
     /protected header/ h'A1010A' /{ \alg\ 1:10 \AES-CCM-16-64-128\}/,
     /unprotected header/ { / iv / 5: h'636898994FF0EC7BFCF6D3F95B'},
     /ciphertext/  h'0573318A3573EB983E55A7C2F06CADD0796C9E584F1D0E3E
                     A8C5B052592A8B2694BE9654F0431F38D5BBC8049FA7F13F'
   ]
  }
}
```

The example above was generated with the key:

```
          h'616263040506070809 0a0b0c0d0e0f10'
```

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 CBOR diagnostic notation) demonstrates such a declaration in the CWT Claims Set of a CWT:

```
{
 /iss/ 1 : "coaps://server.example.com",
 /aud/ 3 : "coaps://client.example.org",
 /exp/ 4 : 1361398824,
 /cnf/ 8 : {
   /kid/ 2 : h'dfd1aa976d8d4575a0fe34b96de2bfad'
  }
}
```

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

Note that the use of a Key ID to identify a proof-of-possesion key needs to be carefully circumscribed, as described below and in Section 6. Where the Key ID is not a cryptographic value derived from the key or where all of the parties involved are not validating the cryptographic derivation, it is possible to get into situations where the same Key ID is being used for multiple keys. The implication of this is that a recipient may have multiple keys known to it that have the same Key ID, and thus it might not know which proof-of-possession key is associated with the CWT.

In the world of constrained Internet of Things (IoT) devices, there is frequently a restriction on the size of Key IDs, either because of table constraints or a desire to keep message sizes small. These restrictions are going to protocol dependent. For example, DTLS can use a Key ID of any size. However, if the key is being used with COSE encrypted message, then the length of the key needs to be minimized and may have a limit as small as one byte.

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

For symmetric keys, the Key ID is normally going to be generated by the CWT issuer. This means that enforcing a rule that Key ID values only match if CWTs have the same issuer works for matching Key IDs between CWTs. In this case, the issuer can ensure that there are no collisions between currently active symmetric keys for all CWTs that it has issued. This allows for a recipient to use the pair of issuer and Key ID for matching keys.

For asymmetric keys, the Key ID value is normally going to be generated by the CWT recipient, thus the possibility of collisions is greater. For instance, recipients might start by assigning a Key ID

of 0, given that Key IDs are frequently only needed to be unique and meaningful to the recipient. This problem can be addressed in a couple of different ways, depending on how the Key ID value is going to be used:

- The issuer can assign a new unique Key ID the first time it sees the key. Depending on the protocol being used, the new value may then need to be transported to the presenter by the protocol used to issue CWTs. In this case, the rule of requiring that the issuer, Key ID pair be used for matching works.

- The issuer can use a different confirmation method if a collision might be unavoidable.

- A recipient can decide not to use a CWT based on a created Key ID if it does not fit the recipient's requirements.

- If an issuer is going to use the Key ID confirmation method and is not going to guarantee that serial number uniqueness is going to be preserved, the recipient needs to have that information configured into it so that appropriate actions can be taken.

3.5. Specifics Intentionally Not Specified

Proof of possession is often demonstrated by having the presenter sign a value determined by the recipient using the key possessed by the presenter. This value is sometimes called a "nonce" or a "challenge".

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

Applications utilizing proof of possession SHOULD also utilize audience restriction, as described in Section 4.1.3 of [JWT], as it provides additional protections. 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.

CBOR Web Tokens with proof-of-possession keys are used in context of an architecture, such as the ACE OAuth Framework [I-D.ietf-ace-oauth-authz], in which protocols are used by a presenter to request these tokens and to subsequently use them with recipients. To avoid replay attacks when the proof-of-possession tokens are sent to presenters, a security protocol, which uses mechansims such as nonces or timestamps, has to be utilized. Note that a discussion of the architecture or specific protocols that CWT proof-of-possession tokens are used with is beyond the scope of this specification.

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

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

5. Privacy Considerations

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

6. Operational Considerations

The use of CWTs with proof-of-possession keys requires additional information to be shared between the involved parties in order to ensure correct processing. The recipient needs to be able to use credentials to verify the authenticity, integrity, and potentially the confidentiality of the CWT and its content. This requires the recipient to know information about the issuer. Likewise, there needs to be agreement between the issuer and the recipient about the claims being used (which is also true of CWTs in general).

When an issuer creates a CWT containing a Key ID claim, it needs to make sure that it does not issue another CWT containing the same Key ID with a different content, or for a different subject, within the lifetime of the CWTs, unless intentionally desired. Failure to do so may allow one party to impersonate another party, with the potential to gain additional privileges. Likewise, if PoP keys are used for multiple different kinds of CWTs in an application and the PoP keys are identified by Key IDs, care must be taken to keep the keys for the different kinds of CWTs segregated so that an attacker cannot cause the wrong PoP key to be used by using a valid Key ID for the wrong kind of CWT.

7. IANA Considerations

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

Values are registered on a Specification Required [RFC8126] basis after a three-week review period on the cwt-reg-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published. [[Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: cwt-reg-review@ietf.org.]]

Registration requests sent to the mailing list for review should use an appropriate subject (e.g., "Request to Register CWT Confirmation Method: example"). Registration requests that are undetermined for a period longer than 21 days can be brought 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.

7.1. CBOR Web Token Claims Registration

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

7.1.1. Registry Contents

- 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]]

7.2. CWT Confirmation Methods Registry

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

7.2.1. Registration Template

Confirmation Method Name:
: The human-readable name requested (e.g., "kid").

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

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

Confirmation Key:
CBOR map key value for the confirmation method.

Confirmation Value Type(s):
CBOR types that can be used for the confirmation method value.

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

7.2.2. 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): COSE_Key structure
- 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): COSE_Encrypt or COSE_Encrypt0 structure (with an optional corresponding 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]]

8. References

8.1. Normative References

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

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

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

8.2. Informative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-21 (work in progress), February 2019.

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

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

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

[OASIS.saml-core-2.0-os] Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005, <http://docs.oasis-open.org/security/saml/v2.0/>.

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

Acknowledgements

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

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

Document History

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

-06

- Corrected nits identified by Roman Danyliw.

-05

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

-04

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

-03

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

-02

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

-01

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

-00

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

Authors' Addresses

Michael B. Jones
Microsoft

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

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@ri.se

Goeran Selander
Ericsson AB
Faeroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Samuel Erdtman
Spotify

Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 11, 2019

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

October 08, 2018

Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE) draft-ietf-ace-dtls-authorize-05

Abstract

This specification defines a profile that allows constrained servers to delegate client authentication and authorization. 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 server can use this protocol to delegate management of authorization information to a trusted host with less severe limitations regarding processing power and memory.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 11, 2019.

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 obtains an access token, bound to a key (the proof-of-possession key), from an authorization server to prove its authorization to access protected resources hosted by the resource server. Also, the client and the resource server are provided by the authorization server with the necessary keying material to establish a DTLS session. The communication between client and authorization server may also be secured with DTLS. This specification supports DTLS with Raw Public Keys (RPK) [RFC7250] and with Pre-Shared Keys (PSK) [RFC4279].

The DTLS handshake [RFC7250] requires the client and server to prove that they can use certain keying material. In the RPK mode, the client proves with the DTLS handshake that it can use the RPK bound to the token and the server shows that it can use a certain RPK. The access token must be presented to the resource server. For the RPK mode, the access token needs to be uploaded to the resource server before the handshake is initiated, as described in Section 5.8.1 of draft-ietf-ace-oauth-authz [1].

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

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].

The authz-info resource refers to the authz-info endpoint as specified in I-D.ietf-ace-oauth-authz [3].

2. Protocol Overview

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

This profile requires the client to retrieve an access token for protected resource(s) it wants to access on RS as specified in I-D.ietf-ace-oauth-authz [4]. Figure 1 shows the typical message flow in this scenario (messages in square brackets are optional):

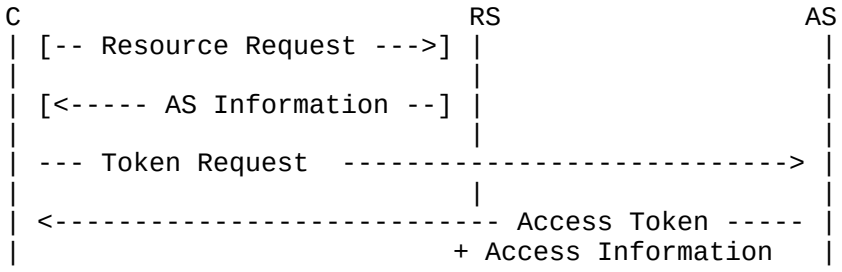


Figure 1: Retrieving an Access Token

To determine the AS in charge of a resource hosted at the RS, C MAY send an initial Unauthorized Resource Request message to the RS. The RS then denies the request and sends an AS information message containing the address of its AS back to the client as specified in Section 5.1.2 of draft-ietf-ace-oauth-authz [5].

Once the client 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 [6]. 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. C may have been registered at the AS via the OAuth 2.0 client registration mechanism as outlined in Section 5.3 of draft-ietf-ace-oauth-authz [7].

The access token returned by the authorization server can then be used by the client to establish a new DTLS session with the resource server. When the client intends to use asymmetric cryptography in the DTLS handshake with the resource server, the client MUST upload the access token to the authz-info resource, i.e. the authz-info endpoint, on the resource server before starting the DTLS handshake, as described in Section 5.8.1 of draft-ietf-ace-oauth-authz [8]. 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 3.3.1).

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.

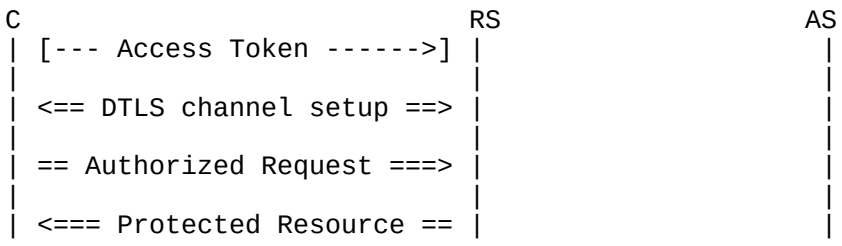


Figure 2: Protocol overview

3. Protocol Flow

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

Section 3.1 describes how the communication between C and AS must be secured. Depending on the used CoAP security mode (see also Section 9 of RFC 7252 [9]), the Client-to-AS request, AS-to-Client response and DTLS session establishment carry slightly different information. Section 3.2 addresses the use of raw public keys while Section 3.3 defines how pre-shared keys are used in this profile.

3.1. Communication between C and AS

To retrieve an access token for the resource that the client wants to access, the client requests an access token from the authorization server. Before C can request the access token, C and AS must establish a secure communication channel. C must securely have obtained keying material to communicate with AS, and C must securely have received authorization information intended for C that states that AS is authorized to provide keying material concerning RS to C. Also, AS must securely have obtained keying material for C, and obtained authorization rules approved by the resource owner (RO) concerning C and RS that relate to this keying material. C and AS must use their respective keying material for all exchanged messages. How the security association between C and AS is established is not part of this document. C and AS MUST ensure the confidentiality, integrity and authenticity of all exchanged messages.

If C is constrained, C and AS should use DTLS to communicate with each other. But C and AS may also use other means to secure their communication, e.g., TLS. The used security protocol must provide

confidentiality, integrity and authenticity, and enable the client to determine if it is the intended recipient of a message, e.g., by using an AEAD mechanism. C must also be able to determine if a response from AS belongs to a certain request. Additionally, the protocol must offer replay protection.

3.2. RawPublicKey Mode

After C and AS mutually authenticated each other and validated each other's authorization, C sends a token request to AS's token endpoint. 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, C MUST use the same keying material that it uses to secure the communication with AS, e.g., the DTLS session.

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

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

Figure 3: Access Token Request Example for RPK Mode

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

AS MUST check if the client that it communicates with is associated with the RPK in the cnf object before issuing an access token to it. If AS determines that the request is to be authorized according to the respective authorization rules, it generates an access token response for C. The response SHOULD contain 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. The

response also contains an access token and an "rs_cnf" parameter containing information about the public key that is used by the resource server. AS MUST ascertain that the RPK specified in "rs_cnf" belongs to the resource server that C wants to communicate with. AS MUST protect the integrity of the token. If the access token contains confidential data, AS MUST also protect the confidentiality of the access token.

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

3.2.1. DTLS Channel Setup Between C and RS

Before the client initiates the DTLS handshake with the resource server, C 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 the RawPublicKey mode, the client MUST specify the public key that AS defined in the "cnf" field of the access token response in the SubjectPublicKeyInfo structure in the DTLS handshake as specified in RFC 7250 [10].

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], [RFC8422]).

Note: According to RFC 7252 [11], CoAP implementations MUST support the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251] and the NIST P-256 curve. As discussed in RFC 7748 [12], 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 RFC 7252 [13].

RS MUST check if the access token is still valid, if RS is the intended destination, i.e., the audience, of the token, and if the token was issued by an authorized AS. The access token is constructed by the authorization server such that the resource server can associate the access token with the Client’s public key. The "cnf" claim MUST contain either C’s RPK or, if the key is already known by the resource server (e.g., from previous communication), a reference to this key. If the authorization server has no certain knowledge that the Client’s key is already known to the resource server, the Client’s public key MUST be included in the access token’s "cnf" parameter. If CBOR web tokens [RFC8392] are used as

recommended in I-D.ietf-ace-oauth-authz [14], unencrypted keys MUST be specified using a "COSE_Key" object, encrypted keys with a "COSE_Encrypt0" structure and references to the key as "key_id" parameters in a CBOR map. RS MUST use the keying material in the handshake that AS specified in the rs_cnf parameter in the access token. Thus, the handshake only finishes if C and RS are able to use their respective keying material.

3.3. 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 shared secret to construct the proof-of-possession token. AS MUST check if the identifier refers to a symmetric key that was previously generated by AS as a shared secret for the communication between this client and the resource server.

The authorization server MUST determine the authorization rules for the C it communicates with as defined by RO and generate the access token accordingly. If the authorization server authorizes the client, it returns an AS-to-Client response. If the profile parameter is present, it is set to "coap_dtls". AS MUST ascertain that the access token is generated for the resource server that C wants to communicate with. Also, AS MUST protect the integrity of the access token. If the token contains confidential data such as the symmetric key, the confidentiality of the token MUST also be protected. Depending on the requested token type and algorithm in the access token request, the authorization server adds access Information to the response that provides the client with sufficient information to setup a DTLS channel with the resource server. AS adds a "cnf" parameter to the access information carrying a "COSE_Key" object that informs the client about the symmetric key that is to be used between C and the resource server.

An example access token response is illustrated in Figure 4. In this example, the authorization server returns a 2.01 response containing a new access token and information for the client, including the symmetric key in the cnf claim. The information is transferred as a CBOR data structure as specified in I-D.ietf-ace-oauth-authz [15].

```
2.01 Created
Content-Format: application/ace+cbor
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 4: Example Access Token Response

The access token also comprises a "cnf" claim. This claim usually contains a "COSE_Key" object that carries either the symmetric key itself or or a key identifier that can be used by the resource server to determine the shared secret. If the access token carries a symmetric key, the access token MUST be encrypted using a "COSE_Encrypt0" structure. The AS MUST use the keying material shared with the RS to encrypt the token.

Instead of providing the keying material, the AS MAY include a key derivation function and a salt in the access token that enables the resource server to calculate the keying material for the communication with C from the access token. In this case, the token contains a "cnf" structure that specifies the key derivation algorithm and the salt that the AS has used to construct the shared key. AS and RS MUST use their shared keying material for the key derivation, and the key derivation MUST follow Section 11 of RFC 8152 [16] with parameters as specified here. The KDF specified in the "alg" parameter SHOULD be HKDF-SHA-256. The salt picked by the AS must be uniformly random and is carried in the "salt" parameter.

The fields in the context information "COSE_KDF_Context" (Section 11.2 of RFC 8152 [17]) MUST have the following values:

- AlgorithmID = "ACE-CoAP-DTLS-salt"

- PartyUInfo = PartyVInfo = (null, null, null)

- keyDataLength is a uint equal the length of the key shared between AS and RS in bits
- protected MUST be a zero length bstr
- other is a zero length bstr
- SuppPrivInfo is omitted

An example "cnf" structure specifying HMAC-based key derivation of a symmetric key with SHA-256 as pseudo-random function and a random salt value is provided in Figure 5.

```
cnf : {
   kty  : symmetric,
   alg  : HKDF-SHA-256,
   salt : h'eIiOFCa9lObw'
}
```

Figure 5: Key Derivation Specification in an Access Token

A response that declines any operation on the requested resource is constructed according to Section 5.2 of RFC 6749 [18], (cf. Section 5.7.3. of draft-ietf-ace-oauth-authz [19]).

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

Figure 6: Example Access Token Response With Reject

3.3.1. DTLS Channel Setup Between C and RS

When a client receives an access token response from an authorization server, C MUST ascertain that the access token response belongs to a certain previously sent access token request, as the request may specify the resource server with which C wants to communicate.

C checks if the payload of the access token response contains an "access_token" parameter and a "cnf" parameter. With this information the client can initiate the establishment of a new DTLS channel with a resource server. To use DTLS with pre-shared keys, the client follows the PSK key exchange algorithm specified in Section 2 of RFC 4279 [20] 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 shared secret by the client and the resource server is used for mutual authentication between both peers. Therefore, the resource server must be able to determine the shared secret from the access token. Following the general ACE authorization framework, the client can upload the access token to the resource server's authz-info resource before starting the DTLS handshake. 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 RFC 7925 [21], 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 key with a matching identifier is found, the resource server MAY process the contents of the "psk_identity" field as access token that is stored with the authorization information endpoint, before continuing the DTLS handshake. If the 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 RFC 7252 [22], 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.

When RS receives an access token, RS MUST check if the access token is still valid, if RS is the intended destination, i.e., the audience of the token, and if the token was issued by an authorized AS. This specification assumes that the access token is a PoP token as described in I-D.ietf-ace-oauth-authz [23] unless specifically stated otherwise. Therefore, the access token is bound to a symmetric PoP key that is used as shared secret between the client and the resource server.

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

3.4. Resource Access

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

With the successful establishment of the DTLS channel, C and RS have proven that they can use their respective keying material. An access token that is bound to the client's keying material is associated with the channel. Any request that the resource server receives on this channel MUST be checked against these authorization rules. RS MUST check for every request if the access token is still valid. 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 [24].

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 client is valid.

3. The request is destined for the resource server.

4. The resource URI specified in the request is covered by the authorization information.

5. The request method is an authorized action on the resource with respect to the authorization information.

Incoming CoAP requests received on a secure DTLS channel that are not thus authorized MUST be rejected according to Section 5.8.2 of draft-ietf-ace-oauth-authz [25]

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 5.1.2 of draft-ietf-ace-oauth-authz [26]) 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.

4. 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 a new access token from the authorization server for the intended action on the respective resource and uploads this access token to the authz-info resource on the resource server.

Figure 7 depicts the message flow where the C requests a new access token after a security association between the client and the resource server has been established using this protocol. If the client wants to update the authorization information, the token request MUST specify the key identifier of the existing DTLS channel between the client and the resource server in the "kid" parameter of the Client-to-AS request. The authorization server MUST verify that the specified "kid" denotes a valid verifier for a proof-of-possession token that has previously been issued to the requesting client. Otherwise, the Client-to-AS request MUST be declined with the error code "unsupported_pop_key" as defined in Section 5.6.3 of draft-ietf-ace-oauth-authz [27].

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.

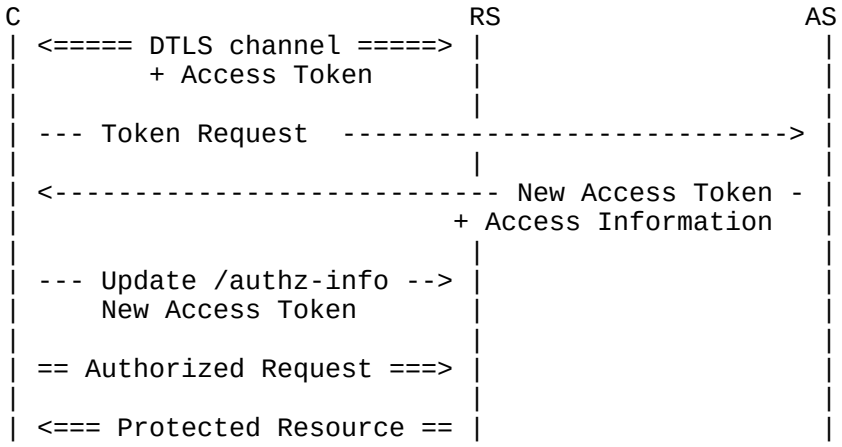


Figure 7: Overview of Dynamic Update Operation

5. 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 MUST terminate existing DTLS sessions after the last valid access token for this session has been deleted.

As specified in Section 5.8.3 of draft-ietf-ace-oauth-authz [28], 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.

Table 1 updates Figure 2 in Section 5.1.2 of draft-ietf-ace-oauth-authz [29] 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

6. 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.

The use of multiple access tokens for a single client increases the strain on the resource server as it must consider every access token and calculate the actual permissions of the client. Also, tokens may contradict each other which may lead the server to enforce wrong permissions. If one of the access tokens expires earlier than others, the resulting permissions may offer insufficient protection. Developers should avoid using multiple access tokens for a client.

7. 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.

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

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

8. 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]

9. References

9.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-16 (work in progress), October 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-02 (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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

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

[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>.

9.2. Informative References

[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>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC8422] Nir, Y., Josefsson, S., and M. Pegourie-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

9.3. URIs

[1] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.8.1

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

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

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

[5] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.1.2

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

[7] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.3

[8] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.8.1

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

[10] https://tools.ietf.org/html/rfc7250

[11] https://tools.ietf.org/html/rfc7252

[12] https://tools.ietf.org/html/rfc7748

[13] https://tools.ietf.org/html/rfc7252

[14] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz

[15] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz

[16] https://tools.ietf.org/html/rfc8152#section-11

[17] https://tools.ietf.org/html/rfc8152#section-11.2

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

[19] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz#section-5.7.3

[20] https://tools.ietf.org/html/rfc4279#section-2

[21] https://tools.ietf.org/html/rfc7925#section-4.2

[22] https://tools.ietf.org/html/rfc7252

[23] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz

[24] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.1.1

[25] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.8.2

[26] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.1.2

[27] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.6.3

[28] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.8.3

[29] https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-16#section-5.1.2

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 AB

Email: goran.selander@ericsson.com

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2019

L. Seitz
RISE
G. Selander
Ericsson
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
Arm Ltd.

February 14, 2019

Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth) draft-ietf-ace-oauth-authz-21

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.

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] 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.

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

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

3. Overview

This specification defines the ACE framework for authorization in the Internet of Things environment. It consists of a set of building blocks.

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

Another building block is the lightweight web transfer protocol CoAP [RFC7252], for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP, which further reduces overhead and message exchanges. While this specification defines extensions for the use of OAuth over CoAP, other underlying protocols are not prohibited from being supported in the future, such as HTTP/2, 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 [RFC8446]). 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.

Refresh Tokens:
Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires, or to obtain additional access tokens with identical or narrower scope (access tokens may have a shorter lifetime and fewer permissions than authorized by the resource owner). Issuing a refresh token is optional at the discretion of the authorization server. If the authorization server issues a refresh token, it is included when issuing an access token (i.e., step (B) in Figure 1).

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

Proof of Possession Tokens:
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 or TLS [RFC6347][RFC8446] [I-D.ietf-tls-dtls13]. CoAP defines a number of proxy operations that require transport layer security to be terminated at the proxy. One approach for protecting CoAP communication end-to-end through proxies, and also to support security for CoAP over a different transport in a uniform way, is to provide security at the application layer using an object-based security mechanism such as COSE [RFC8152].

One application of COSE is OSCORE [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 for use in constrained environments.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the token endpoint and optionally the introspection endpoint. A client obtains an access token, and optionally a refresh token, from an AS using the token endpoint and subsequently presents the access token to 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].

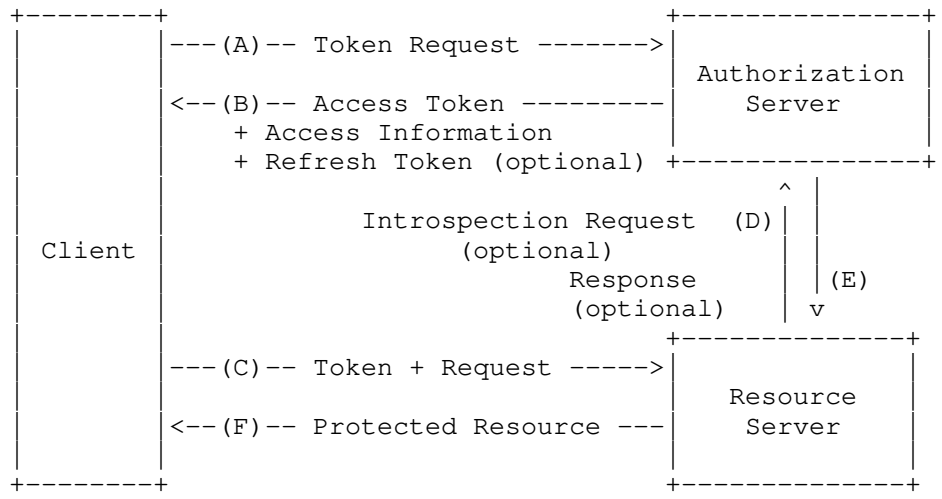


Figure 1: Basic Protocol Flow.

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 and optionally a refresh token (note that only certain grant types support refresh tokens). It can also return additional parameters, referred to as "Access Information". In addition to the response parameters defined by OAuth 2.0 and the PoP access token extension, this framework defines parameters that can be used to inform the client about capabilities of the RS. 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 Access Information obtained from the AS.

The Client and the RS mutually authenticate using the security protocol specified in the profile (see step B) and the keys obtained in the access token or the Access Information. The RS verifies that the token is integrity protected 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, integrity protected and protected against message replay. It is also REQUIRED that the AS and the endpoint communicating with it (client or RS) perform mutual

authentication. Furthermore it MUST be assured that responses are bound to the requests in the sense that the receiver of a response can be certain that the response actually belongs to a certain request.

Profiles MUST specify a communication security protocol that provides the features required above.

In OAuth 2.0 the communication with the Token and the Introspection endpoints at the AS is assumed to be via HTTP and may use Uri-query parameters. When profiles of this framework use CoAP instead, 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. Profiles that use CBOR encoding of protocol message parameters MUST use the media format 'application/ace+cbor', unless the protocol message is wrapped in another Content-Format (e.g. object security). If CoAP is used for communication, the Content-Format MUST be abbreviated with the ID: 19 (see Section 8.15.

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, other discovery methods may be used, or the client may be pre-provisioned with the address of the AS.

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 Request Creation Hints 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 Request Creation Hints

The AS Request Creation Hints message is sent by RS as a response to an Unauthorized Resource Request message (see Section 5.1.1) to help the sender of the Unauthorized Resource Request message in acquiring a valid access token. The AS Request Creation Hints message is CBOR map, with a MANDATORY element "AS" specifying an absolute URI (see Section 4.3 of [RFC3986]) that identifies the AS in charge of RS.

The message can also contain the following OPTIONAL parameters:

- A "audience" element containing a suggested audience that the client should request towards the AS.

- A "kid" element containing the key identifier of a key used in an existing security association between the client and the RS. The RS expects the client to request an access token bound to this key, in order to avoid having to re-establish the security association.

- A "nonce" element containing a nonce generated by RS to ensure freshness in case that the RS and AS do not have synchronized clocks.

- A "scope" element containing the suggested scope that the client should request towards the AS.

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

Name	CBOR Key	Value Type
AS	0	text string
kid	4	byte string
nonce	5	byte string
scope	9	text or byte string
audience	18	text string

Figure 2: AS Request Creation Hints

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

Figure 3 shows an example for an AS Request Creation Hints message payload using CBOR [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",
 audience: "coaps://rs.example.com",
 nonce: h'e0a156bb3f',
 scope: "rTempC"
}
```

Figure 3: AS Request Creation Hints 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 Request Creation Hints 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.

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 Request Creation Hints 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 [RFC6749] defines four grant types. The grant types can be split up into two groups, those granted on behalf of the resource owner (password, authorization code, implicit) and those for the client (client credentials). Further grant types have been added later, such as [RFC7521] defining an assertion-based authorization grant.

The grant type is selected depending on the use case. In cases where the client acts on behalf of the resource owner, 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 how the communication is protected. The content of the request consists of the parameters specified in Section 4 of the OAuth 2.0 specification [RFC6749] with the exception of the "grant_type" parameter, which is OPTIONAL in the context of this framework (as opposed to REQUIRED in RFC6749). If that parameter is missing, the default value "client_credentials" is implied.

Furthermore the "audience" parameter from [I-D.ietf-oauth-token-exchange] can be used to request an access token bound to a specific audience.

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

If CBOR is used then this parameter MUST be encoded as a CBOR map. The "scope" parameter can be formatted as specified in [RFC6749] and additionally as a byte string, in order to allow compact encoding of complex scopes.

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.

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

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

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

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

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

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
OSCORE: 0x19, 0x05, 0x05, 0x44, 0x61, 0x6c, 0x65, 0x6b
Content-Format: "application/oscore"
Payload:
  0x44025d1 ... (full payload omitted for brevity) ... 68b3825e
  )


Decrypted payload:
{
  "client_id" : "myclient",
  "req_cnf" : {
    "COSE_Key" : {
      "kty" : "EC",
      "kid" : h'11',
      "crv" : "P-256",
      "x" : b64'usWxHK2PmfnHKwXPS54m0kTcGJ90UiglWiGahtagnv8',
      "y" : b64'IBOL+C3BttVivg+lSreASjpkttcsz+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 the client performs a password based authentication in this example by submitting its client_secret (see Section 2.3.1 of [RFC6749]). Note that this example uses the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "client_id" : "myclient",
  "client_secret" : "mysecret234",
  "audience" : "valve424",
  "scope" : "read",
  "req_cnf" : {
    "kid" : b64'6kg0dXJM13U'
  }
}
```

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

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

5.6.2. 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 Access Information. When using CBOR payloads, the content MUST be encoded as CBOR map, containing parameters as specified in Section 5.1 of [RFC6749], with the following additions and changes:

profile:

OPTIONAL. This indicates the profile that the client MUST use towards the RS. See Section 5.6.4.3 for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

token_type:
This parameter is OPTIONAL, as opposed to 'required' in [RFC6749]. By default implementations of this framework SHOULD assume that the token_type is "pop". If a specific use case requires another token_type (e.g., "Bearer") to be used then this parameter is REQUIRED.

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

Figure 8 summarizes the parameters that may be part of the Access Information. This does not include the additional parameters specified in [I-D.ietf-ace-oauth-params].

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]

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key, which is defined in [I-D.ietf-ace-oauth-params].

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (remainder of CWT omitted for brevity;
   CWT contains COSE_Key in the "cnf" claim)',
  "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:

- When using CBOR 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 content type (for CoAP-based interactions) or media type (for HTTP-based interactions) "application/ace+cbor" MUST be used for the error response.
- 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	1
invalid_client	2
invalid_grant	3
unauthorized_client	4
unsupported_grant_type	5
invalid_scope	6
unsupported_pop_key	7
incompatible_profiles	8

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.

- If the client and the RS it has requested an access token for do not share a common profile, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "incompatible_profiles" 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. 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

5.6.4.2. Token Type

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

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

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

In this framework the "pop" value for the "token_type" parameter is the default. The AS may, however, provide a different value.

5.6.4.3. Profile

Profiles of this framework MUST define the communication protocol and the communication security protocol between the client and the RS. The security protocol MUST provide encryption, integrity and replay protection. It MUST also provide a binding between requests and responses. Furthermore profiles MUST define 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 Access Information in the access token response in order to support negotiation or signaling of profile specific parameters.

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 the abbreviations corresponding to claims with the abbreviations defined in [RFC8392].

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

Name	CBOR Key	Value Type
access_token	1	byte string
scope	9	text or byte string
audience	18	text string
client_id	24	text string
client_secret	25	byte string
response_type	26	text string
redirect_uri	27	text string
state	28	text string
code	29	byte string
error	30	unsigned integer
error_description	31	text string
error_uri	32	text string
grant_type	33	unsigned integer
token_type	34	unsigned integer
expires_in	35	unsigned integer
username	36	text string
password	37	text string
refresh_token	38	byte string
profile	39	unsigned integer

Figure 12: CBOR mappings used in token requests

5.7. The Introspection 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 requesting entity and the introspection endpoint at the AS MUST be integrity protected and encrypted. The communication security protocol MUST also provide a binding between requests and responses. Furthermore the two interacting parties MUST perform mutual authentication. Finally the AS SHOULD verify that the requesting entity has the right to access introspection information about the provided token. Profiles of this framework that support introspection MUST specify how authentication and communication security between the requesting entity and the AS is implemented.

The default name of this endpoint in an url-path 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. Introspection Request

The requesting entity sends a POST request to the introspection endpoint at the AS, the profile MUST specify how the communication is protected. If 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 requesting entity to aid the AS in its response MAY be included.

For CoAP-based interaction, all messages MUST use the content type "application/ace+cbor", while for HTTP-based interactions the equivalent media type "application/ace+cbor" MUST be used.

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 OSCORE [I-D.ietf-core-object-security] is assumed in this example, therefore the Content-Format is "application/oscore". Figure 14 shows the decoded payload.

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

Figure 13: Example introspection request.

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

Figure 14: Decoded token.

5.7.2. Introspection Response

If the introspection request is authorized and successfully processed, the AS sends a response with the response code equivalent to the CoAP code 2.01 (Created). If the introspection request was invalid, not authorized or couldn't be processed the AS returns an error response as described in Section 5.7.3.

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 addition:

profile OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.6.4.3 for more details on the formatting of this parameter.

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

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

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

Figure 15: 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 and CBOR is used the payload MUST be encoded as a CBOR map and the Content-Format "application/ace+cbor" MUST be used.

- If the credentials used by the requesting entity (usually the RS) are invalid the AS MUST respond with the response code equivalent to the CoAP code 4.01 (Unauthorized) and use the required and optional parameters from Section 5.2 in RFC 6749 [RFC6749].

- If the requesting entity does not have the right to perform this introspection request, the AS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden). In this case no payload is returned.

- The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12.

- The error codes MUST be abbreviated using the codes specified in 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 16, using the given integer abbreviation for the map key.

Note that we have aligned abbreviations that correspond to a claim with the abbreviations defined in [RFC8392] and the abbreviations of parameters with the same name from Section 5.6.5.

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
scope	9	text or byte string
active	10	True or False
token	12	byte string
client_id	24	text string
error	30	unsigned integer
error_description	31	text string
error_uri	32	text string
token_type_hint	33	text string
token_type	34	text string
username	36	text string
profile	39	unsigned integer

Figure 16: 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 document uses the "cnf" claim from

[I-D.ietf-ace-cwt-proof-of-possession] and specifies the "scope" claim for JWT- and CWT-encoded 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 strings 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 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 5.6.4.3.

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). Section Section 5.8.1.1 outlines how an RS MUST proceed to verify the validity of an access token.

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

This specification RECOMMENDS that an RS stores only one token per proof-of-possession key, meaning that an additional token linked to the same key will overwrite any existing token at the RS.

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

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

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

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

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

5.8.1.1. Verifying an Access Token

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

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

In case the token is communicated by reference the RS needs to obtain the claims first. When the RS uses token introspection the relevant specification is [RFC7662] with CoAP transport specified in Section 5.7.

Errors may happen during this initial processing stage:

- If token or claim verification fails, the RS MUST discard the token and, if this was an interaction with authz-info, return an error message with a response code equivalent to the CoAP code 4.01 (Unauthorized).

- If the claims cannot be obtained the RS MUST discard the token and, in case of an interaction via the authz-info endpoint, return an error message with a response code equivalent to the CoAP code 4.00 (Bad Request).

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

iss The issuer claim must identify an AS that has the authority to issue access tokens for the receiving RS. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.01 (Unauthorized).

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

aud The audience claim must refer to an audience that the RS identifies with. If that is not the case the RS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden).

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

If the access token contains any other claims that the RS cannot process the RS MUST respond with a response code equivalent to the CoAP code 4.00 (Bad Request). The RS MAY provide additional detail in the error response to clarify which claim couldn't be processed.

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

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

5.8.1.2. Protecting the Authorization Information Endpoint

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

Specifically it SHOULD NOT be possible to perform GET, DELETE or PUT on the authz-info endpoint and on it's children (if any).

The POST method SHOULD NOT be allowed on children of the authz-info endpoint.

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

5.8.2. Client Requests to the RS

If an RS receives a request from a client, and the target resource requires authorization, the RS MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession 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 expiration of a received access token. Here follows a list of the possibilities including what functionality they require of the RS.

- The token is a CWT and includes an "exp" claim and possibly the "nbf" claim. The RS verifies these by comparing them to values from its internal clock as defined in [RFC7519]. In this case the RS's internal clock must reflect the current date and time, or at least be synchronized with the AS's clock. How this clock synchronization would be performed is out of scope for this specification.

- The RS verifies the validity of the token by performing an introspection request as specified in Section 5.7. 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).

- In order to support token expiration for devices that have no reliable way of synchronizing their internal clocks, this specification defines the following approach: The claim "exi" ("expires in") can be used, to provide the RS with the lifetime of the token in seconds from the time the RS first receives the token. This approach is of course vulnerable to malicious clients holding back tokens they do not want to expire. Such an attack can only be prevented if the RS is able to communicate with the AS in some regular intervals, so that the can AS provide the RS with a list of expired tokens. The drawback of this mitigation is that the RS might as well use the communication with the AS to synchronize its internal clock.

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

5.8.4. Key Expriation

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

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

- The client knows a default validity period for all tokens it is using. This information could be provisioned to the client when it is registered at the AS, or published by the AS in a way that the client can query.

- The AS informs the client about the token validity using the "expires_in" parameter in the Access Information.

- The client performs an introspection of the token. Although this is not explicitly forbidden, how exactly this is done is not currently specified for OAuth.

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

6. Security Considerations

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

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.

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

It is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. 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. This risk can also be mitigated to some extent by making sure that keys are refreshed more frequently.

If clients are capable of doing so, they should frequently request fresh access tokens, as this allows the AS to keep the lifetime of the tokens short. This allows the AS to use shorter proof-of-possession key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens, they SHOULD scope these access tokens to a specific permission.

6.1. Unprotected AS Request Creation Hints

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

6.2. Minimal security requirements for communication

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

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

RS-AS The communication between the Resource Server and the Authorization Server via the introspection endpoint MUST be encrypted, integrity and replay protected. Furthermore responses from the AS to the RS MUST be bound to the RS's request. This requires that the client and the Authorization Server have previously exchanged either a shared secret, or their public keys in order to negotiate a secure communication. Furthermore the RS MUST be able to determine whether an AS has the authority to issue access tokens itself. This is usually configured out of band, but could also be performed through an online lookup mechanism provided that it is also secured in the same way.

C-RS The initial communication between the client and the Resource Server can not be secured in general, since the RS is not in possession of on access token for that client, which would carry the necessary parameters. Certain security mechanisms (e.g. DTLS with server-side authentication via a certificate or a raw public key) can be possible and are RECOMMEND if supported by both parties. After the client has successfully transmitted the access token to the RS, a secure communication protocol MUST be

established between client and RS for the actual resource request. This protocol MUST provide encryption, integrity and replay protection as well as a binding between requests and responses. This requires that the client learned either the RS's public key or received a symmetric proof-of-possession key bound to the access token from the AS. The RS must have learned either the client's public key or a shared symmetric key from the claims in the token or an introspection request. Since ACE does not provide profile negotiation between C and RS, the client MUST have learned what profile the RS supports (e.g. from the AS or pre-configured) and initiate the communication accordingly.

6.3. Use of Nonces for Replay Protection

The RS may add a nonce to the AS Request Creation Hints 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.

6.4. Combining profiles

There may be use cases were different profiles of this framework are combined. For example, an MQTT-TLS profile is used between the client and the RS in combination with a CoAP-DTLS profile for interactions between the client and the AS. Ideally, profiles should be designed in a way that the security of system should not depend on the specific security mechanisms used in individual protocol interactions.

6.5. Unprotected Information

Communication with the authz-info endpoint, as well as the various error responses defined in this framework all potentially include sending information over an unprotected channel. These messages may leak information to an adversary. 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.

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

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

If the initial unauthorized resource request message (see Section 5.1.1) is used, the client MUST make sure that it is not sending sensitive content in this request. While GET and DELETE requests only reveal the target URI of the resource, while POST and PUT requests would reveal the whole payload of the intended operation.

6.6. Identifying audiences

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

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

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

Even the client must be able to determine the correct values to put into the "audience" parameter, in order to obtain a token for the intended RS. Errors in this process can lead to the client inadvertently communicating with the wrong RS. The correct values for "audience" can either be provisioned to the client as part of its configuration, or provided by the RS as part of the "AS Request Creation Hints" Section 5.1.2 or dynamically looked up by the client

in some directory. In the latter case the integrity and correctness of the directory data must be assured.

6.7. Denial of service against or with Introspection

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

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

Second an attacker could use the fact that an RS performs introspection to perform a denial of service attack against that RS by repeatedly sending tokens to its authz-info endpoint that require an introspection call. RS can mitigate such attacks by implementing a rate limit on how many introspection requests they perform in a given time interval and rejecting incoming requests to authz-info for a certain amount of time, when that rate limit has been reached.

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 (see 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. ACE Authorization Server Request Creation Hints

This specification establishes the IANA "ACE Authorization Server Request Creation Hints" registry. The registry has been created to use the "Expert Review 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 Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry defined in [RFC6749].

- Error name: "unsupported_pop_key"
- Error usage location: AS token endpoint error response
- Related protocol extension: The ACE framework [this document]
- Change Controller: IESG
- Specification document(s): Section 5.6.3 of [this document]

- Error name: "incompatible_profiles"
- Error usage location: AS token endpoint error response
- Related protocol extension: The ACE framework [this document]
- Change Controller: IESG
- Specification document(s): Section 5.6.3 of [this document]

8.3. OAuth Error Code CBOR Mappings Registry

This specification establishes the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review 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.

8.4. OAuth Grant Type CBOR Mappings

This specification establishes the IANA "OAuth Grant Type CBOR Mappings" registry. The registry has been created to use the "Expert Review 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.

8.5. 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]

8.6. OAuth Access Token Type CBOR Mappings

This specification established the IANA "OAuth Access Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review 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.

8.6.1. Initial Registry Contents

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

- Name: "pop"
- Value: 2
- Reference: [this document]
- Original Specification: [this document]

8.7. ACE Profile Registry

This specification 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.

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

8.8. OAuth Parameter Registration

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

- Name: "profile"
- Parameter Usage Location: token response
- Change Controller: IESG
- Reference: Section 5.6.4.3 of [this document]

8.9. OAuth Parameters CBOR Mappings Registry

This specification establishes the IANA "OAuth Parameters CBOR Mappings" registry. The registry has been created to use the "Expert Review 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 parameter 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 the mappings of parameters corresponding to claim names intentionally coincide with the CWT claim name mappings from [RFC8392].

8.10. OAuth Introspection Response Parameter Registration

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

- 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]

8.11. OAuth Token Introspection Response CBOR Mappings Registry

This specification establishes the IANA "OAuth Token Introspection Response CBOR Mappings" registry. The registry has been created to use the "Expert Review 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 16. The Reference column for all of these entries will be this document.

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

8.12. 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]

- Claim Name: "profile"
- Claim Description: The profile a token is supposed to be used with.
- Change Controller: IESG
- Reference: Section 5.8 of [this document]

- Claim Name: "exi"
- Claim Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker from of token expiration for devices that cannot synchronize their internal clocks.
- Change Controller: IESG
- Reference: Section 5.8.3 of [this document]

8.13. CBOR Web Token Claims

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

- Claim Name: "scope"
- Claim Description: The scope of an access token as defined in [RFC6749].
- JWT Claim Name: scope
- Claim Key: TBD (suggested: 9)
- Claim Value Type(s): byte string or text string
- Change Controller: IESG
- Specification Document(s): Section 5.8 of [this document]

- Claim Name: "profile"
- Claim Description: The profile a token is supposed to be used with.
- JWT Claim Name: profile
- Claim Key: TBD (suggested: 39)
- Claim Value Type(s): integer
- Change Controller: IESG
- Specification Document(s): Section 5.8 of [this document]

- Claim Name: "exi"
- Claim Description: The expiration time of a token measured from when it was received at the RS in seconds.
- JWT Claim Name: exi
- Claim Key: TBD (suggested: 41)
- Claim Value Type(s): integer
- Change Controller: IESG
- Specification Document(s): Section 5.8 of [this document]

8.14. Media Type Registrations

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

Type name: application

Subtype name: ace+cbor

Required parameters: none

Optional parameters: none

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

Security considerations: See Section 6 of this document.

Interoperability considerations: n/a

Published specification: [this document]

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

Additional information:

Magic number(s): n/a

File extension(s): .ace

Macintosh file type code(s): n/a

Person & email address to contact for further information: Ludwig Seitz <ludwig.seitz@ri.se>

Intended usage: COMMON

Restrictions on usage: None

Author: Ludwig Seitz <ludwig.setiz@ri.se>

Change controller: IESG

8.15. CoAP Content-Format Registry

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

Media Type: application/ace+cbor

Encoding

ID: 19

Reference: [this document]

8.16. Expert Review Instructions

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

Expert reviewers should take into consideration the following points:

- Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered, and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.
- Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

- Since a high degree of overlap is expected between these registries and the contents of the OAuth parameters [IANA.OAuthParameters] registries, experts should require new registrations to maintain a reasonable level of alignment with parameters from OAuth that have comparable functionality.

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.

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

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., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-05 (work in progress), November 2018.

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-04 (work in progress), February 2019.

[I-D.ietf-oauth-token-exchange]
Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", draft-ietf-oauth-token-exchange-16 (work in progress), October 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>.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
Framework: Bearer Token Usage", RFC 6750,
DOI 10.17487/RFC6750, October 2012,
<https://www.rfc-editor.org/info/rfc6750>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

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

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

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

[I-D.ietf-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-14 (work in progress), January 2019.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-30 (work in progress), November 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), August 2010.

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

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

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

[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>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8516] Keraenen, A., ""Too Many Requests" Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <https://www.rfc-editor.org/info/rfc8516>.

Appendix A. Design Justification

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

Common IoT constraints are:

Low Power Radio:

Many IoT devices are equipped with a small battery which needs to last for a long time. For many constrained wireless devices, the highest energy cost is associated to transmitting or receiving messages (roughly by a factor of 10 compared to AES) [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 RECOMMENDS 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.

Access Information:

This framework defines the name "Access Information" for data concerning the RS that the AS returns to the client in an access token response (see Section 5.6.2). 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 then (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 [RFC8414]
* Optionally: Handle refresh tokens.

Client

* Discover the AS in charge of the RS that is to be targeted with a request.
* Submit the token request (see step (A) of Figure 1).
    + Authenticate to the AS.
    + Optionally (if not pre-configured): Specify which RS, which resource(s), and which action(s) the request(s) will target.
    + If raw public keys (rpk) or certificates are used, make sure the AS has the right rpk or certificate for this client.
* Process the access token and Access Information (see step (B) of Figure 1).
    + Check that the Access Information provides the necessary security parameters (e.g., PoP key, information on communication security protocols supported by the RS).
    + Safely store the proof-of-possession key.

    + If provided by the AS: Safely store the refresh token.
  * Send the token and request to the RS (see step (C) of Figure 1).

    + Authenticate towards the RS (this could coincide with the proof of possession process).
    + Transmit the token as specified by the AS (default is to the authz-info endpoint, alternative options are specified by profiles).
    + Perform the proof-of-possession procedure as specified by the profile in use (this may already have been taken care of through the authentication procedure).
  * Process the RS response (see step (F) of Figure 1) of the RS.

Resource Server

  * Expose a way to submit access tokens. By default this is the authz-info endpoint.
  * Process an access token.

    + Verify the token is from a recognized AS.
    + 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.
  * Safely store credentials such as raw public keys for authentication or proof-of-possession keys linked to access tokens.

Appendix C. Requirements on Profiles

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

- Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.6.4.3
- 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 5.6.4.3
- Specify how the client and the RS mutually authenticate. Section 4
- Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol. Section 5.6.4.2
- Specify a unique profile identifier. Section 5.6.4.3
- 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. This must provide encryption, integrity protection, replay protection and a binding between requests and responses. Section 5 and 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).
- Whether the RS has synchronized time (and thus is able to use the 'exp' claim) or not.

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 17.

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 Access Information. The PoP access token contains the public key of the client, and the Access Information contains the public key of the RS. For communication security this example uses DTLS RawPublicKey between the client and the RS. The issued token will have a short

validity time, i.e., "exp" close to "iat", 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.

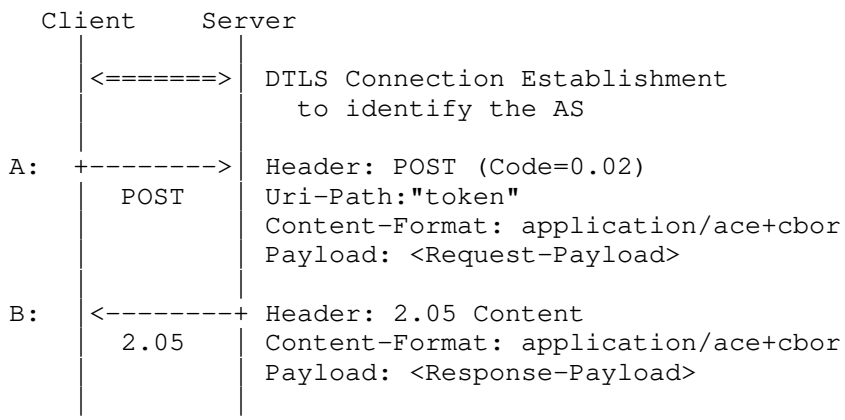


Figure 17: Token Request and Response Using Client Credentials.

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

```
Request-Payload :
{
  "audience" : "tempSensorInLivingRoom",
  "client_id" : "myclient",
  "client_secret" : "qwerty"
  "req_cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}

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

Figure 18: Request and Response Payload Details.

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

```
{
  "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 19: Access Token including Public Key of the Client.

Messages C and F are shown in Figure 20 - Figure 21.

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

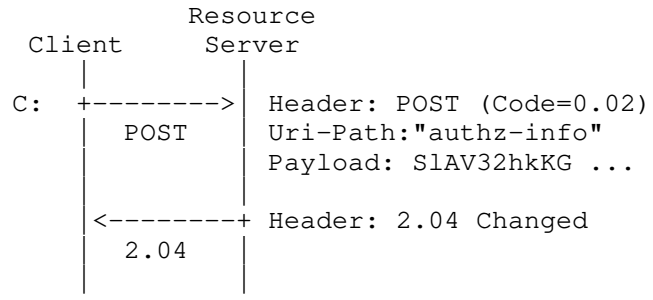


Figure 20: Access Token provisioning to RS

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

The client sends the CoAP request GET to /temperature on RS over DTLS. The RS verifies that the request is authorized, based on previously established security context.

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

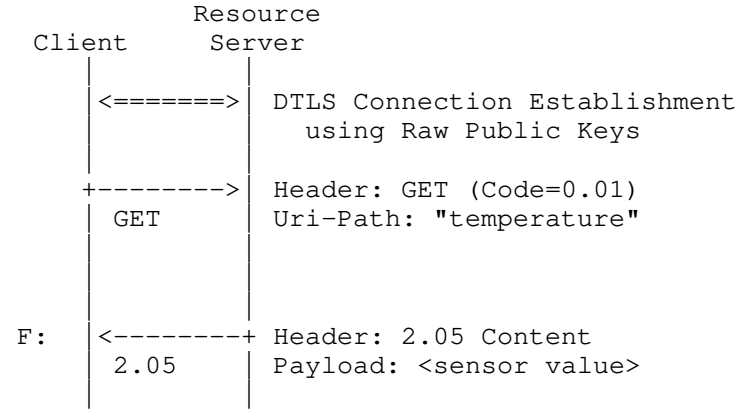


Figure 21: Resource Request and Response protected by DTLS.

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 22.

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

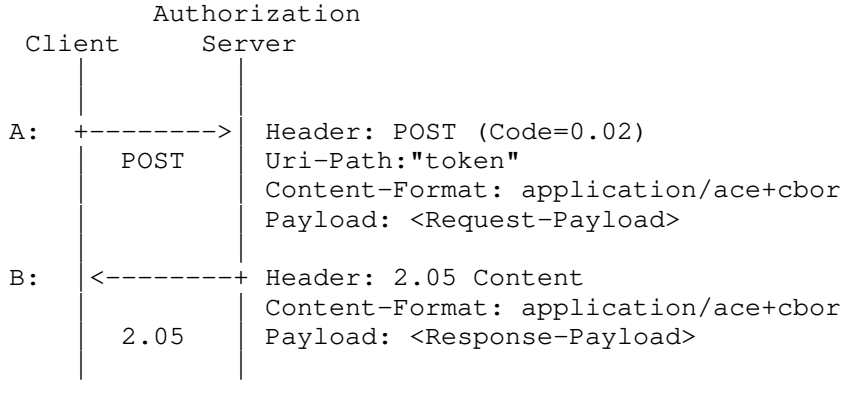


Figure 22: Token Request and Response using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 23.

```
Request-Payload:
{
  "client_id" : "keyfob",
  "client_secret" : "qwerty"
}

Response-Payload:
{
  "access_token" : b64'VGVzdCB0b2tlbg==',
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "oct",
      "alg" : "HS256",
      "k": b64'ZoRSOrFzN_FzUA5XKMYoVHyzff5oRJxl-IXRtztJ6uE'
    }
  }
}
```

Figure 23: Request and Response Payload for C offline

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

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

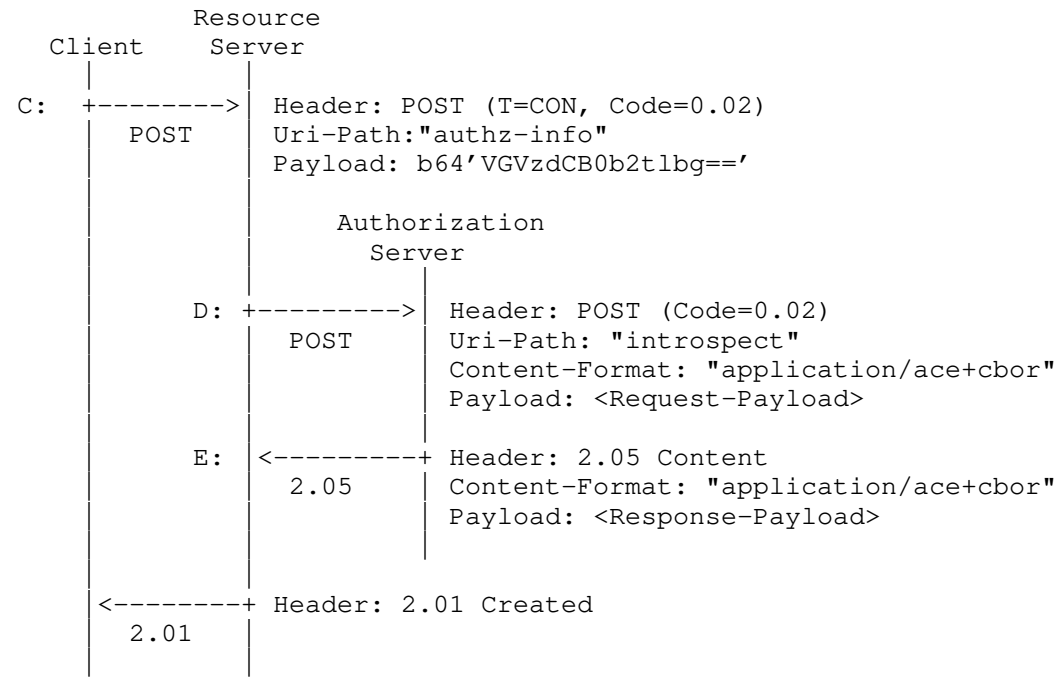


Figure 24: Token Introspection for C offline

The information contained in the Request-Payload and the Response-Payload is shown in Figure 25.

```
Request-Payload:
{
  "token" : b64'VGVzdCB0b2tlbg==',
  "client_id" : "FrontDoor",
  "client_secret" : "ytrewq"
}

Response-Payload:
{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1311280970,
  "cnf" : {
    "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk'
  }
}
```

Figure 25: Request and Response Payload for Introspection

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

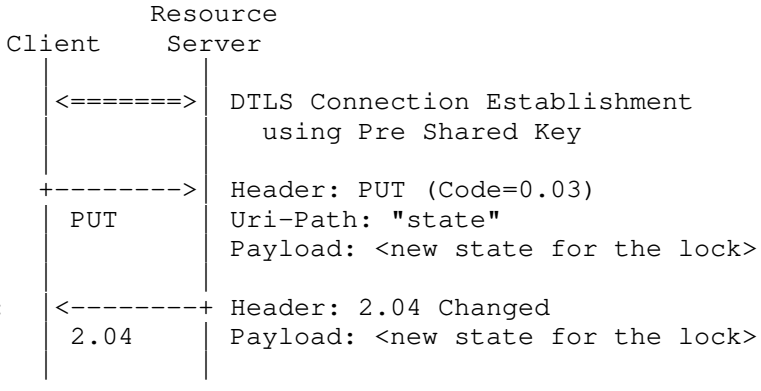


Figure 26: Resource request and response protected by OSCORE

Appendix F. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

F.1. Verion -20 to 21

- Added text about expiration of RS keys.

F.2. Verion -19 to 20

- Replaced "req_aud" with "audience" from the OAuth token exchange draft.
- Updated examples to remove unnecessary elements.

F.3. Version -18 to -19

- Added definition of "Authorization Information".
- Explicitly state that ACE allows encoding refresh tokens in binary format in addition to strings.
- Renamed "AS Information" to "AS Request Creation Hints" and added the possibility to specify req_aud and scope as hints.
- Added the "kid" parameter to AS Request Creation Hints.
- Added security considerations about the integrity protection of tokens with multi-RS audiences.

- Renamed IANA registries mapping OAuth parameters to reflect the mapped registry.
- Added JWT claim names to CWT claim registrations.
- Added expert review instructions.
- Updated references to TLS from 1.2 to 1.3.

F.4. Version -17 to -18

- Added OSCORE options in examples involving OSCORE.
- Removed requirement for the client to send application/cwt, since the client has no way to know.
- Clarified verification of tokens by the RS.
- Added exi claim CWT registration.

F.5. Version -16 to -17

- Added references to (D)TLS 1.3.
- Added requirement that responses are bound to requests.
- Specify that grant_type is OPTIONAL in C2AS requests (as opposed to REQUIRED in OAuth).
- Replaced examples with hypothetical COSE profile with OSCORE.
- Added requirement for content type application/ace+cbor in error responses for token and introspection requests and responses.
- Reworked abbreviation space for claims, request and response parameters.
- Added text that the RS may indicate that it is busy at the authz-info resource.
- Added section that specifies how the RS verifies an access token.
- Added section on the protection of the authz-info endpoint.
- Removed the expiration mechanism based on sequence numbers.
- Added reference to RFC7662 security considerations.
- Added considerations on minimal security requirements for communication.
- Added security considerations on unprotected information sent to authz-info and in the error responses.

F.6. Version -15 to -16

- Added text the RS using RFC6750 error codes.
- Defined an error code for incompatible token request parameters.
- Removed references to the actors draft.
- Fixed errors in examples.

F.7. Version -14 to -15

- Added text about refresh tokens.
- Added text about protection of credentials.
- Rephrased introspection so that other entities than RS can do it.

- Editorial improvements.

F.8. Version -13 to -14

- Split out the 'aud', 'cnf' and 'rs_cnf' parameters to [I-D.ietf-ace-oauth-params]
- Introduced the "application/ace+cbor" Content-Type.
- Added claim registrations from 'profile' and 'rs_cnf'.
- Added note on schema part of AS Information Section 5.1.2
- Realigned the parameter abbreviations to push rarely used ones to the 2-byte encoding size of CBOR integers.

F.9. Version -12 to -13

- Changed "Resource Information" to "Access Information" to avoid confusion.
- Clarified section about AS discovery.
- Editorial changes

F.10. 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.11. Version -10 to -11

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

F.12. 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.13. Version -08 to -09

- Allowed scope to be byte strings.
- 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.14. 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.15. Version -06 to -07

- Various clarifications added.
- Fixed erroneous author email.

F.16. 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.17. 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.18. 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.19. 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.20. 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.21. 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.

- o 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
- o Appendix B Added list of roles and responsibilities for C, AS and RS.

Authors' Addresses

Ludwig Seitz
RISE
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.
Absam 6067
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 23, 2019

F. Palombini
Ericsson AB
L. Seitz
RISE SICS AB
G. Selander
Ericsson AB
M. Gunnarsson
RISE SICS AB

February 19, 2019

OSCORE profile of the Authentication and Authorization for Constrained Environments Framework
draft-ietf-ace-oscore-profile-07

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.

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].

OSCORE specifies how to use CBOR Object Signing and Encryption (COSE) [RFC8152] to secure CoAP messages. 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", "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].

RESTful terminology follows HTTP [RFC7231].

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749], 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.

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

2. Protocol Overview

This section gives an overview on how to use the ACE Framework [I-D.ietf-ace-oauth-authz] to secure the communication between a client and a resource server using OSCORE [I-D.ietf-core-object-security]. The parameters needed by the client to negotiate the use of this profile with the authorization server, as well as OSCORE setup process, are described in detail in the following sections.

This profile requires a client to retrieve an access token from the AS for the resource it wants to access on a RS, using the token endpoint, as specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

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 as specified in section 5.1 of [I-D.ietf-ace-oauth-authz]. The access token request and response MUST be confidentiality-protected and ensure authenticity. This profile RECOMMENDS the use of OSCORE between client and AS, but TLS or DTLS MAY be used additionally or instead.

Once the client has retrieved the access token, it generates a nonce N1 and posts both the token and N1 to the RS using the authz-info endpoint and mechanisms specified in section 5.8 of [I-D.ietf-ace-oauth-authz] and Content-Format = application/ace+cbor.

If the access token is valid, the RS replies to this request with a 2.01 (Created) response with Content-Format = application/ace+cbor, which contains a nonce N2 in a CBOR map. Moreover, the server concatenates N1 with N2 and sets the ID Context in the Security Context (see section 3 of [I-D.ietf-core-object-security]) to N1 concatenated with N2. The RS then derives the complete Security Context associated with the received token from it plus the parameters received in the AS, following section 3.2 of [I-D.ietf-core-object-security].

After receiving the nonce N2, the client concatenates it with N1 and sets the ID Context in its Security Context (see section 3 of [I-D.ietf-core-object-security]) to N1 concatenated with N2. The client then derives the complete Security Context from the ID Context plus the parameters received from the AS.

Finally, the client sends a request protected with OSCORE to the RS. This message may contain the ID Context value. If the request verifies, then this Security Context is stored in the server, and used in the response, and in further communications with the client, until token expiration. This Security Context is discarded if the same token is re-used to successfully derive a new Security Context. Once the client receives a valid response, it does not continue to include the ID Context value in further requests.

The use of random nonces during the exchange prevents the reuse of AEAD nonces and keys with different messages, in case of re-derivation of the Security Context both for Clients and Resource Servers from an old non-expired access token, e.g. in case of re-boot of either the client or RS. In fact, by using random nonces as ID Context, the request to the authz-info endpoint posting the same token results in a different Security Context, since Master Secret, Sender ID and Recipient ID are the same but ID Context is different. Therefore, the main requirement for the nonces is that they have a

good amount of randomness. If random nonces were not used, a node re-using a non-expired old token would be susceptible to on-path attackers provoking the creation of OSCORE messages using old AEAD keys and nonces.

An overview of the profile flow for the OSCORE profile is given in Figure 1.

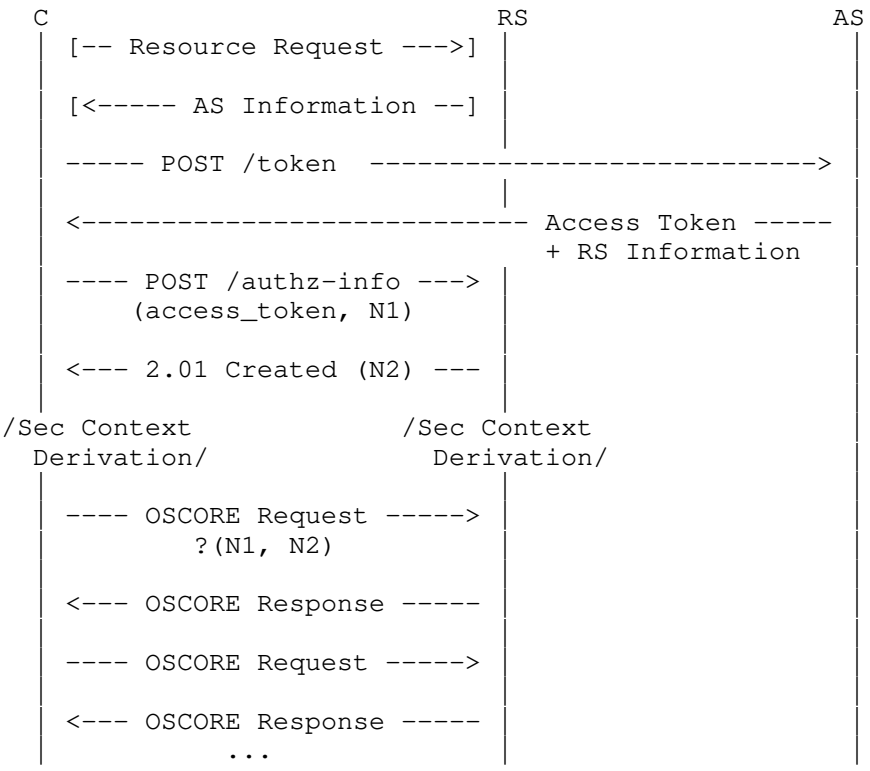


Figure 1: Protocol Overview

3. Client-AS Communication

The following subsections describe the details of the POST request and response to the token endpoint between client and AS. Section 3.2 of [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, which the client receives from the AS in this exchange. The proof-of-possession key (pop-key) provisioned from the AS MUST be used as master secret in OSCORE.

3.1. C-to-AS: POST to token endpoint

The client-to-AS request is specified in Section 5.6.1 of [I-D.ietf-ace-oauth-authz].

The client MUST send this POST request to the token endpoint over a secure channel that guarantees authentication, message integrity and confidentiality (see Section 5).

An example of such a request, in CBOR diagnostic notation without the tag and value abbreviations is reported in Figure 2

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

Figure 2: Example C-to-AS POST /token request for an access token bound to a symmetric key.

If the client wants to update its access rights without changing an existing OSCORE Security Context, it MUST include in its POST request to the token endpoint a req_cnf object carrying the client's identifier (that was assigned in section Section 3.2) in the kid field. This identifier can be used by the AS to determine the shared secret to construct the proof-of-possession token and therefore MUST identify a symmetric key that was previously generated by the AS as a shared secret for the communication between the client and the RS. The AS MUST verify that the received value identifies a proof-of-possession key and token that have previously been issued to the requesting client. If that is not the case, the Client-to-AS request MUST be declined with the error code 'invalid_request' as defined in Section 5.6.3 of [I-D.ietf-ace-oauth-authz].

An example of such a request, in CBOR diagnostic notation without the tag and value abbreviations is reported in Figure 3

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

Figure 3: Example C-to-AS POST /token request for updating rights to an access token bound to a symmetric key.

3.2. AS-to-C: Access Token

After verifying the POST request to the token endpoint and that the client is authorized to obtain an access token corresponding to its access token request, the AS responds as defined in section 5.6.2 of [I-D.ietf-ace-oauth-authz]. If the client request was invalid, or not authorized, the AS returns an error response as described in section 5.6.3 of [I-D.ietf-ace-oauth-authz].

The AS 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 4.3 to derive the security context to run OSCORE. Usually it is assumed that constrained devices will be pre-configured with the necessary profile, so that this kind of profile negotiation can be omitted.

Moreover, the AS MUST provision the following data:

- a master secret
- a server identifier

Additionally, the AS MAY provision the following data, in the same response.

- a client identifier
- an AEAD algorithm
- an HKDF algorithm

- a salt

- a replay window type and size

The master secret MUST be communicated as the 'ms' field in the OSCORE_Security_Context in the 'cnf' parameter of the access token response as defined in Section 3.2 of [I-D.ietf-ace-oauth-params]. The OSCORE_Security_Context is a CBOR map object, defined in Section 3.2.1. The AEAD algorithm MAY be included as the 'alg' parameter in the OSCORE_Security_Context; the HKDF algorithm MAY be included as the 'hkdf' parameter of the OSCORE_Security_Context, the salt MAY be included as the 'salt' parameter of the COSCORE_Security_Context, and the replay window type and size MAY be included as the 'rpl' of the OSCORE_Security_Context, as defined in Section 3.2.1.

The same parameters MUST be included as metadata of the access token. This profile RECOMMENDS the use of CBOR web token (CWT) as specified in [RFC8392]. If the token is a CWT, the same OSCORE_Security_Context structure defined above MUST be placed in the 'cnf' claim of this token.

The AS MUST also assign an identifier to the RS, and MAY assign an identifier to the client. These identifiers 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]. The client identifiers MUST be unique in the set of all clients on a single RS, and RS identifiers MUST be unique in the set of all RS for any given client. Moreover, when assigned, these MUST be included in the OSCORE_Security_Context, as defined in Section 3.2.1.

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 in Section 4.3 C sets the Sender ID of its Security Context to the clientId value received and the Recipient ID to the serverId value, and RS does the opposite.

Figure 4 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/ace+cbor"
Payload:
{
  "access_token" : h'a5037674656d7053656e73 ...'
   (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600",
  "cnf" : {
    "OSCORE_Security_Context" : {
      "alg" : "AES-CCM-16-64-128",
      "clientId" : b64'qA',
      "serverId" : b64'Qg',
      "ms" : h'f9af838368e353e78888e1426bd94e6f'
    }
  }
}
```

Figure 4: Example AS-to-C Access Token response with OSCORE profile.

Figure 5 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" : {
    "OSCORE_Security_Context" : {
      "alg" : "AES-CCM-16-64-128",
      "clientId" : "client",
      "serverId" : "server",
      "ms" : h'f9af838368e353e78888e1426bd94e6f'
  }
}
```

Figure 5: Example CWT with OSCORE parameters.

The same CWT token as in Figure 5, using the value abbreviations defined in [I-D.ietf-ace-oauth-authz] and [I-D.ietf-ace-cwt-proof-of-possession] and encoded in CBOR is shown in Figure 6.

NOTE TO THE RFC EDITOR: before publishing, it should be checked (and in case fixed) that the values used below (which are not yet registered) are the final values registered in IANA.

```
A5                                      # map(5)
   03                                   # unsigned(3)
   76                                   # text(22)
      74656D7053656E736F72496E4C6976696E67526F6F6D
                                        # "tempSensorInLivingRoom"
   06                                   # unsigned(6)
   1A 5112D728                          # unsigned(1360189224)
   04                                   # unsigned(4)
   1A 51145DC8                          # unsigned(1360289224)
   09                                   # unsigned(9)
   78 18                                # text(24)
      74656D7065726174757265F6720666972 6D776172655F70
                                        # "temperature_g firmware_p"
   08                                   # unsigned(8)
   A1                                   # map(1)
      04                                # unsigned(4)
      A4                                # map(4)
         05                             # unsigned(5)
         0A                             # unsigned(10)
         02                             # unsigned(2)
         66                             # text(6)
            636C69656E74                # "client"
         03                             # unsigned(3)
         66                             # text(6)
            736572766572                # "server"
         01                             # unsigned(1)
         50                             # bytes(16)
            F9AF838368E353E78888E1426BD94E6F
```

Figure 6: Example CWT with OSCORE parameters.

If the client has requested an update to its access rights using the same OSCORE Security Context, which is valid and authorized, the AS MUST omit the 'cnf' parameter in the response, and MUST carry the client identifier in the 'kid' field in the 'cnf' parameter of the token. The client identifier needs to be provisioned, in order for the RS to identify the previously generated Security Context.

Figure 7 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/ace+cbor"
Payload:
{
  "access_token" : h'a5037674656d7053656e73 ...'
   (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600"
}
```

Figure 7: Example AS-to-C Access Token response with OSCORE profile, for update of access rights.

Figure 8 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim for update of access rights, in CBOR diagnostic notation without tag and value abbreviations.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_h",
  "cnf" : {
    "kid" : b64'qA'
  }
}
```

Figure 8: Example CWT with OSCORE parameters for update of access rights.

3.2.1. OSCORE_Security_Context Object

An OSCORE_Security_Context is an object that represents part or all of an OSCORE Security Context (Section 3.1 of [I-D.ietf-core-object-security]). The OSCORE_Security_Context object can either be encoded as JSON or as CBOR. In both cases, the set of common parameters that can appear in an OSCORE_Security_Context object can be found in the IANA "OSCORE Security Context Parameters" registry (Section Section 9.2) and is defined below. All parameters are optional. Table 1 provides a summary of the OSCORE_Security_Context parameters defined in this section.

name	CBOR label	CBOR type	registry	description
ms	1	bstr		OSCORE Master Secret value
clientId	2	bstr		OSCORE Sender ID value of the client, OSCORE Recipient ID value of the server
serverId	3	bstr		OSCORE Sender ID value of the server, OSCORE Recipient ID value of the client
hkdf	4	bstr / int	COSE Algorithm Values (HMAC-based)	OSCORE HKDF value
alg	5	tstr / int	COSE Algorithm Values (AEAD)	OSCORE AEAD Algorithm value
salt	6	bstr		OSCORE Master Salt value
contextId	7	bstr		OSCORE ID Context value
rpl	8	bstr / int		OSCORE Replay Window Type and Size

Table 1: OSCORE_Security_Context Parameters

ms: This parameter identifies the OSCORE Master Secret value, which is a byte string. For more information about this field, see

section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "ms" value is a Base64 encoded byte string. In CBOR, the "ms" type is bstr, and has label 1.

clientId: This parameter identifies a client identifier as a byte string. This identifier is used as OSCORE Sender ID in the client and OSCORE Recipient ID in the server. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "clientId" value is a Base64 encoded byte string. In CBOR, the "clientId" type is bstr, and has label 2.

serverId: This parameter identifies a server identifier as a byte string. This identifier is used as OSCORE Sender ID in the server and OSCORE Recipient ID in the client. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "serverId" value is a Base64 encoded byte string. In CBOR, the "serverId" type is bstr, and has label 3.

hkdf: This parameter identifies the OSCORE HKDF Algorithm. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. The values used MUST be registered in the IANA "COSE Algorithms" registry and MUST be HMAC-based HKDF algorithms. The value can either be the integer or the text string value of the HMAC-based HKDF algorithm in the "COSE Algorithms" registry. In JSON, the "hkdf" value is a case-sensitive ASCII string or an integer. In CBOR, the "hkdf" type is tstr or int, and has label 4.

alg: This parameter identifies the OSCORE AEAD Algorithm. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security] The values used MUST be registered in the IANA "COSE Algorithms" registry and MUST be AEAD algorithms. The value can either be the integer or the text string value of the HMAC-based HKDF algorithm in the "COSE Algorithms" registry. In JSON, the "alg" value is a case-sensitive ASCII string or an integer. In CBOR, the "alg" type is tstr or int, and has label 5.

salt: This parameter identifies the OSCORE Master Salt value, which is a byte string. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "salt" value is a Base64 encoded byte string. In CBOR, the "salt" type is bstr, and has label 6.

contextId: This parameter identifies the security context as a byte string. This identifier is used as OSCORE ID Context. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "contextID" value

is a Base64 encoded byte string. In CBOR, the "contextID" type is bstr, and has label 7.

rpl: This parameter is used to carry the OSCORE value, encoded as a bstr. This parameter identifies the OSCORE Replay Window Size and Type value, which is a byte string. For more information about this field, see section 3.1 of [I-D.ietf-core-object-security]. In JSON, the "rpl" value is a Base64 encoded byte string. In CBOR, the "rpl" type is bstr, and has label 8.

An example of JSON OSCORE_Security_Context is given in Figure 9.

```
"OSCORE_Security_Context" : {
  "alg" : "AES-CCM-16-64-128",
  "clientId" : b64'qA',
  "serverId" : b64'Qg',
  "ms" : b64'+a+Dg2jjU+eIiOFCa9lObw'
}
```

Figure 9: Example JSON OSCORE_Security_Context object

The CDDL grammar describing the CBOR OSCORE_Security_Context object is:

```
OSCORE_Security_Context = {
    ? 1 => bstr,              ; ms
    ? 2 => bstr,              ; clientId
    ? 3 => bstr,              ; serverId
    ? 4 => tstr / int,        ; hkdf
    ? 5 => tstr / int,        ; alg
    ? 6 => bstr,              ; salt
    ? 7 => bstr,              ; contextId
    ? 8 => bstr / tstr,       ; rpl
    * int / tstr => any
}
```

4. Client-RS Communication

The following subsections describe the details of the POST request and response to the authz-info endpoint between client and RS. The client generates a nonce N1 and posts it together with the token that includes the materials provisioned by the AS to the RS. The RS then derives a nonce N2 and use Section 3.2 of [I-D.ietf-core-object-security] to derive a security context based on a shared master secret and the two nonces, established between client and server.

Note that the proof-of-possession required to bind the access token to the client is implicitly performed by generating the shared OSCORE Security Context using the pop-key as master secret, for both client and RS. An attacker using a stolen token will not be able to generate a valid OSCORE context and thus not be able to prove possession of the pop-key.

4.1. C-to-RS: POST to authz-info endpoint

The client MUST generate a nonce N1 very unlikely to have been previously used with the same input keying material. This profile RECOMMENDS to use a 64-bit long random number as nonce. The client MUST store this nonce as long as the response from the RS is not received and the access token related to it is still valid. The client MUST use CoAP and the Authorization Information resource as described in section 5.8.1 of [I-D.ietf-ace-oauth-authz] to transport the token and N1 to the RS. The client MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [I-D.ietf-ace-oauth-authz]. The client MUST include the access token using the correct CBOR label (e.g., "cwt" for CWT, "jwt" for JWT) and N1 using the 'nonce' parameter defined in section 5.1.2 of [I-D.ietf-ace-oauth-authz].

The authz-info endpoint is not protected, nor are the responses from this resource.

The access token MUST be encrypted, since it is transferred from the client to the RS over an unprotected channel.

Note that a client may be required to re-POST the access token, since an RS may delete a stored access token, due to lack of memory.

Figure 10 shows an example of the request sent from the client to the RS, in CBOR diagnostic notation without the tag and value abbreviations.

```
Header: POST (Code=0.02)
Uri-Host: "rs.example.com"
Uri-Path: "authz-info"
Content-Format: "application/ace+cbor"
Payload:
  {
    "access_token": h'a5037674656d7053656e73 ...'
  (remainder of access token omitted for brevity)',
    "nonce": h'018a278f7faab55a'
  }
```

Figure 10: Example C-to-RS POST /authz-info request using CWT

4.2. RS-to-C: 2.01 (Created)

The RS MUST follow the procedures defined in section 5.8.1 of [I-D.ietf-ace-oauth-authz]: the RS MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with 2.01 (Created). If the token is valid but is associated to claims that the RS cannot process (e.g., an unknown scope), or if any of the expected parameters in the OSCORE_Security_Context is missing (e.g. any of the mandatory parameters from the AS), or if any parameters received in the OSCORE_Security_Context is unrecognized, the RS MUST respond with an error response code equivalent to the CoAP code 4.00 (Bad Request). In the latter two cases, 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.

Additionally, the RS MUST generate a nonce N2 very unlikely to have been previously used with the same input keying material, and send it within the 2.01 (Created) response. The payload of the 2.01 (Created) response MUST be a CBOR map containing the 'nonce' parameter defined in section 5.1.2 of [I-D.ietf-ace-oauth-authz], set to N2. This profile RECOMMENDS to use a 64-bit long random number as nonce. Moreover, if the OSCORE_Security_Context in the token did not contain a 'clientId' parameter, the RS MUST generate an identifier, unique in the set of all its existing client identifiers, and send it in a 'clientId' parameter in the CBOR map as a CBOR bstr. The RS MAY generate and send a 'ClientId' identifier even though the OSCORE_Security_Context contained such a parameter, in order to guarantee the uniqueness of the client identifier. The RS MUST use the Content-Format "application/ace+cbor" defined in section 8.14 of [I-D.ietf-ace-oauth-authz].

Figure 11 shows an example of the response sent from the RS to the client, in CBOR diagnostic notation without the tag and value abbreviations.

```
Header: Created (Code=2.01)
Content-Format: "application/ace+cbor"
Payload:
  {
    "nonce": h'25a8991cd700ac01'
  }
```

Figure 11: Example RS-to-C 2.01 (Created) response

When receiving an updated access token with updated authorization information from the client (see section Section 3.1), it is RECOMMENDED that the RS overwrites the previous token, that is only the latest authorization information in the token received by the RS is valid. This simplifies for the RS to keep track of authorization information for a given client.

As specified in section 5.8.3 of [I-D.ietf-ace-oauth-authz], the RS MUST notify the client with an error response with code 4.01 (Unauthorized) for any long running request before terminating the session, when the access token expires.

4.3. OSCORE Setup

Once receiving the 2.01 (Created) response from the RS, following the POST request to authz-info endpoint, the client MUST extract the nonce N2 from the 'nonce' parameter and the client identifier from the 'clientId' in the CBOR map in the payload of the response. Then, the client MUST set the ID Context of the Security Context created to communicate with the RS to the concatenation of N1 and N2, in this order: ID Context = N1 | N2, where | denotes byte string concatenation. The client MUST set the Master Secret and Recipient ID from the parameters received from the AS in Section 3.2. The client MUST set the AEAD Algorithm, Master Salt, HKDF, and Replay Window from the parameters received from the AS in Section 3.2, if present. In case these parameters are omitted, the default values are used as described in section 3.2 of [I-D.ietf-core-object-security]. The client MUST set the Sender ID from the 'clientId in the 2.01 (Created) response, if present; otherwise, the client MUST set the Sender ID from the parameters received from the AS in Section 3.2. After that, the client MUST derive the complete Security Context following section 3.2.1 of [I-D.ietf-core-object-security]. From this point on, the client MUST

use this Security Context to communicate with the RS when accessing the resources as specified by the authorization information.

If any of the expected parameters is missing (e.g. any of the mandatory parameters from the AS, or the 'clientId', either received from the AS or in the 2.01 (Created) response from the RS), the client MUST stop the exchange, and MUST NOT derive the Security Context. The client MAY restart the exchange, to get the correct security material.

The client then uses this Security Context to send requests to RS using OSCORE. In the first request sent to the RS, the client MAY include the kid context if the application needs to, with value ID Context, i.e. N1 concatenated with N2.

After sending the 2.01 (Created) response, the RS MUST set the ID Context of the Security Context created to communicate with the client to the concatenation of N1 and N2, in this order: ID Context = N1 | N2, where | denotes byte string concatenation. The RS MUST set the Master Secret, Sender ID and Recipient ID from the parameters, received from the client in the access token in Section 4.1 after validation of the token as specified in Section 4.2. The RS MUST set the AEAD Algorithm, Master Salt, HKDF, and Replay Window from the parameters received from the client in the access token in Section 4.1 after validation of the token as specified in Section 4.2, if present. In case these parameters are omitted, the default values are used as described in section 3.2 of [I-D.ietf-core-object-security]. After that, the RS MUST derive the complete Security Context following section 3.2.1 of [I-D.ietf-core-object-security], and MUST associate this Security Context with the authorization information from the access token.

The RS then uses this Security Context to verify the request and send responses to C using OSCORE. If OSCORE verification fails, error responses are used, as specified in section 8 of [I-D.ietf-core-object-security]. Additionally, if OSCORE verification succeeds, the verification of access rights is performed as described in section Section 4.4. The RS MUST NOT use the Security Context after the related token has expired, and MUST respond with a unprotected 4.01 (Unauthorized) error message.

4.4. Access rights verification

The RS MUST follow the procedures defined in section 5.8.2 of [I-D.ietf-ace-oauth-authz]: if an RS receives an OSCORE-protected request from a client, then the RS processes it according to [I-D.ietf-core-object-security]. If OSCORE verification succeeds, and the target resource requires authorization, the RS retrieves the

authorization information from the access token associated to the Security Context. The RS then MUST verify that the authorization information covers the resource and the action requested.

The response code MUST be 4.01 (Unauthorized) in case the client has not used the Security Context associated with the access token, 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).

5. Secure Communication with AS

As specified in the ACE framework (section 5.7 of [I-D.ietf-ace-oauth-authz]), the requesting entity (RS and/or client) and the AS communicates via the introspection or token endpoint. The use of CoAP and OSCORE for this communication is RECOMMENDED in this profile, other protocols (such as HTTP and DTLS or TLS) MAY be used instead.

If OSCORE is used, the requesting entity 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 requesting entity and the AS communicate using OSCORE ([I-D.ietf-core-object-security]) through the introspection endpoint as specified in section 5.7 of [I-D.ietf-ace-oauth-authz] and through the token endpoint as specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

6. Discarding the Security Context

There are a number of scenarios where a client or RS needs to discard the OSCORE security context, and acquire a new one.

The client MUST discard the current security context associated with an RS when:

- the Sequence Number space ends.
- the access token associated with the context expires.
- the client receives a number of 4.01 Unauthorized responses to OSCORE requests using the same security context. The exact number needs to be specified by the application.

- the client receives a new nonce in the 2.01 (Created) response (see Section 4.2) to a POST request to the authz-info endpoint, when re-posting a non-expired token associated to the existing context.

The RS MUST discard the current security context associated with a client when:

- Sequence Number space ends.
- Access token associated with the context expires.

7. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. Thus the general security considerations from the framework also apply to this profile.

Furthermore the general security considerations of OSCORE [I-D.ietf-core-object-security] also apply to this specific use of the OSCORE protocol.

OSCORE is designed to secure point-to-point communication, providing a secure binding between the request and the response(s). Thus the basic OSCORE protocol is not intended for use in point-to-multipoint communication (e.g. multicast, publish-subscribe). Implementers of this profile should make sure that their usecase corresponds to the expected use of OSCORE, to prevent weakening the security assurances provided by OSCORE.

Since the use of nonces in the exchange guarantees uniqueness of AEAD keys and nonces, it is REQUIRED that nonces are not reused with the same input keying material even in case of re-boots. This document RECOMMENDS the use of 64 bit random nonces to guarantee non-reuse; if applications use something else, such as a counter, they need to guarantee that reboot and lost of state on either node does not provoke re-use. If that is not guaranteed, nodes are still susceptible to re-using AEAD nonces and keys, in case the Security Context is lost, and on-path attacker replay messages.

This profiles recommends that the RS maintains a single access token for a client. The use of multiple access tokens for a single client increases the strain on the resource server as it must consider every access token and calculate the actual permissions of the client. Also, tokens may contradict each other which may lead the server to enforce wrong permissions. If one of the access tokens expires earlier than others, the resulting permissions may offer insufficient

protection. Developers should avoid using multiple access tokens for a client.

8. Privacy Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. Thus the general privacy considerations from the framework also apply to this profile.

As this document uses OSCORE, thus the privacy considerations from [I-D.ietf-core-object-security] apply here as well.

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 an OSCORE Security Context already exists between the client and the resource server, more detailed information may be included.

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

9. IANA Considerations

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

9.1. ACE OAuth Profile Registry

The following registration is done for the ACE OAuth Profile Registry following the procedure specified in section 8.7 of [I-D.ietf-ace-oauth-authz]:

- Profile name: coap_oscore
- Profile Description: Profile for using OSCORE to secure communication between constrained nodes using the Authentication and Authorization for Constrained Environments framework.
- Profile ID: TBD (value between 1 and 255)
- Change Controller: IESG
- Specification Document(s): [[this specification]]

9.2. OSCORE Security Context Parameters Registry

It is requested that IANA create a new registry entitled "OSCORE Security Context Parameters" registry. The registry is to be created as Expert Review Required. Guidelines for the experts is provided Section 9.5. It should be noted that in addition to the expert review, some portions of the registry require a specification, potentially on standards track, be supplied as well.

The columns of the registry are:

name The JSON name requested (e.g., "ms"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short. This name 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. The name is not used in the CBOR encoding.

CBOR label The value to be used to identify this algorithm. Key map labels MUST be unique. The label can be a positive integer, a negative integer or a string. Integer values between 0 and 255 and strings of length 1 are designated as Standards Track Document required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values of 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.

CBOR Type This field contains the CBOR type for the field.

registry This field denotes the registry that values may come from, if one exists.

description This field contains a brief description for the field.

specification This contains a pointer to the public specification for the field if one exists

This registry will be initially populated by the values in Table 1. The specification column for all of these entries will be this document.

9.3. CWT Confirmation Methods Registry

The following registration is done for the CWT Confirmation Methods Registry following the procedure specified in section 7.2.1 of [I-D.ietf-ace-cwt-proof-of-possession]:

- Confirmation Method Name: "OSCORE_Security_Context"
- Confirmation Method Description: OSCORE_Security_Context carrying the OSCORE Security Context parameters
- Confirmation Key: TBD (value between 4 and 255)

- Confirmation Value Type(s): map
- Change Controller: IESG
- Specification Document(s): Section 3.2.1 of [[this specification]]

9.4. JWT Confirmation Methods Registry

The following registration is done for the JWT Confirmation Methods Registry following the procedure specified in section 6.2.1 of [RFC7800]:

- Confirmation Method Value: "osc"
- Confirmation Method Description: OSCORE_Security_Context carrying the OSCORE Security Context parameters
- Change Controller: IESG
- Specification Document(s): Section 3.2.1 of [[this specification]]

9.5. Expert Review Instructions

The IANA registry established in this document is defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments, code points in other ranges should not be assigned for testing.
- Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be

used on, and the number of code points left that encode to that size.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-21 (work in progress), February 2019.

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-04 (work in progress), February 2019.

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

[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>.

[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.ietf-ace-cwt-proof-of-possession]
Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", draft-ietf-ace-cwt-proof-of-possession-05 (work in progress), November 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>.

[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>.

[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. 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/ace+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

Acknowledgments

The authors wish to thank Jim Schaad and Marco Tiloca for the input on this memo.

Authors' Addresses

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
RISE SICS AB
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Martin Gunnarsson
RISE SICS AB
Scheelevagen 17
Lund 22370
Sweden

Email: martin.gunnarsson@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 29, 2018

F. Palombini
Ericsson

June 27, 2018

CoAP Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE) draft-palombini-ace-coap-pubsub-profile-03

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on December 29, 2018.

Copyright Notice

Table of Contents

1. Introduction

The publisher-subscriber setting allows for devices with limited reachability to communicate via a broker that enables store-and-forward messaging between the devices. The pub-sub scenario using the Constrained Application Protocol (CoAP) is specified in [I-D.ietf-core-coap-pubsub]. This document defines a way to authorize nodes in a CoAP pub-sub type of setting, using the ACE framework [I-D.ietf-ace-oauth-authz].

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz], [I-D.palombini-ace-key-groupcomm] and [I-D.ietf-core-coap-pubsub]. In particular, analogously to [I-D.ietf-ace-oauth-authz], terminology for entities in the architecture such as Client (C), Resource Server (RS), and Authorization Server (AS) is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors], and terminology for entities such as the Key Distribution Center (KDC) and Dispatcher in [I-D.palombini-ace-key-groupcomm].

2. Profile Overview

The objective of this document is to specify how to protect a CoAP pub-sub communication, as described in [I-D.ietf-core-coap-pubsub], using [I-D.palombini-ace-key-groupcomm], which itself expands the Ace framework ([I-D.ietf-ace-oauth-authz]), and profiles ([I-D.ietf-ace-dtls-authorize], [I-D.ietf-ace-oscore-profile]).

The architecture of the scenario is shown in Figure 1.

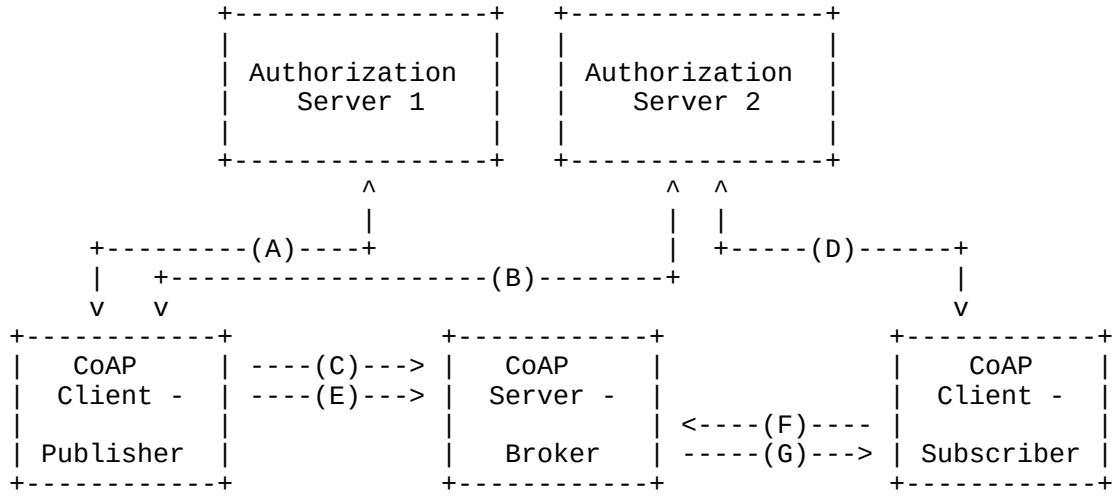


Figure 1: Architecture CoAP pubsub with Authorization Servers

The RS is the broker, which contains the topic. This node corresponds to the Dispatcher, in [I-D.palombini-ace-key-groupcomm]. The AS1 hosts the policies about the Broker: what endpoints are allowed to Publish on the Broker. The Clients access this node to get write access to the Broker. The AS2 hosts the policies about the topic: what endpoints are allowed to access what topic. This node represents both the AS and Key Distribution Center roles from [I-D.palombini-ace-key-groupcomm].

There are four phases, the first three can be done in parallel.

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

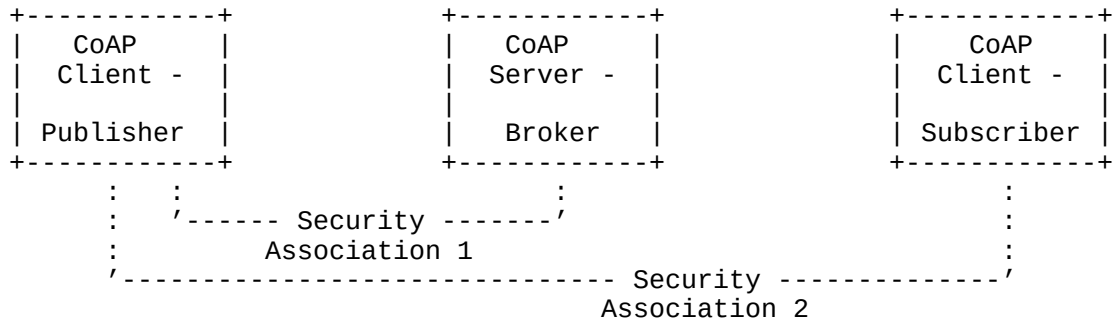
2. The Publisher requests access to a specific topic at the AS2

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

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

This exchange aims at setting up 2 different security associations: on the one hand, the Publisher has a security association with the Broker, to protect the communication and securely authorize the Publisher to publish on a topic (Security Association 1). On the other hand, the Publisher has a security association with the Subscriber, to protect the publication content itself (Security Association 2). The Security Association 1 is set up using AS1 and a profile of [I-D.ietf-ace-oauth-authz], the Security Association 2 is set up using AS2 and [I-D.palombini-ace-key-groupcomm].

Note that, analogously to the Publisher, the Subscriber can also set up an additional security association with the Broker, using an AS, in the same way the Publisher does with AS1. In this case, only authorized Subscribers would be able to get notifications from the Broker. The overhead would be that each Subscriber should access the AS and get all the information to start a secure exchange with the Broker.



Note that AS1 and AS2 might either be co-resident or be 2 separate physical entities, in which case access control policies must be exchanged between AS1 and AS2, so that they agree on rights for joining nodes about specific topics. How the policies are exchanged is out of scope for this profile.

3. coap_pubsub Profile

This profile uses [I-D.palombini-ace-key-groupcomm], which expands the ACE framework. This document specifies which exact parameters from [I-D.palombini-ace-key-groupcomm] have to be used, and the values for each parameter.

The Publisher and the Subscriber map to the Client in [I-D.palombini-ace-key-groupcomm], the AS2 maps to the AS and to the KDC, the Broker maps to the Dispatcher.

Note that both publishers and subscribers use the same profile, called "coap_pubsub".

3.1. Retrieval of COSE Key for protection of content

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

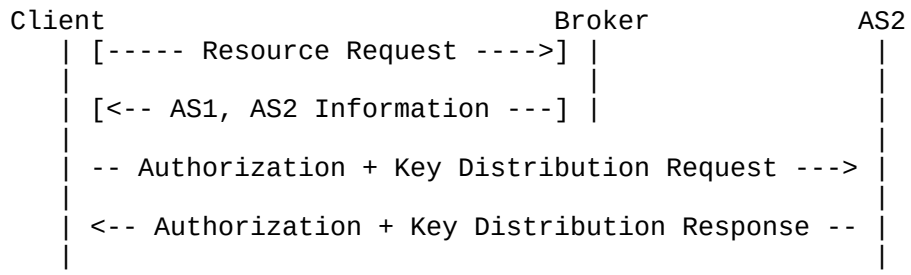


Figure 2: B: Access request - response

Complementary to what is defined in [I-D.ietf-ace-oauth-authz] (Section 5.1.1), to determine the AS2 in charge of a topic hosted at the Broker, the Broker MAY send the address of both the AS in charge of the topic back to the Client in the 'AS' parameter in the AS Information, as a response to an Unauthorized Resource Request (Section 5.1.2). An example using CBOR diagnostic notation is given below:

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

Figure 3: AS1, AS2 Information example

After retrieving the AS2 address, the Client sends an Authorization + Key Distribution Request, which is an Authorization Request merged with a Key Distribution Request, as described in [I-D.palombini-ace-key-groupcomm], Sections 3.1 and 4.1. The reason for merging these two messages is that the AS2 is both the AS and the KDC, in this setting, so the Authorization Response and the Post Token message are not necessary.

More specifically, the Client sends a POST request to the /token endpoint on AS2, that MUST contain in the payload (formatted as a CBOR map):

- the following fields from the Authorization Request (Section 3.1 of [I-D.palombini-ace-key-groupcomm]):
  - the grant type set to "client_credentials",
  - OPTIONALLY, if needed, other additional parameters such as "Client_id"
- the following fields from the Key Distribution Request (Section 4.1 of [I-D.palombini-ace-key-groupcomm]):
  - the client_cred parameter containing the Client's public key, if the Client needs to directly send that to the AS2,
  - the scope parameter set to a CBOR array containing the broker's topic as first element and the string "publisher" for publishers and "subscriber" for subscribers as second element
  - the get_pub_keys parameter set to the empty array if the Client needs to retrieve the public keys of the other pubsub members
  - OPTIONALLY, if needed, the pub_keys_repos parameters

Note that the alg parameter in the client_cred COSE_Key MUST be a signing algorithm, as defined in section 8 of [RFC8152].

Examples of the payload of a Authorization + Key Distribution Request are specified in Figure 5 and Figure 8.

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

The AS2 response is an Authorization + Key Distribution Response, see Section 4.2 of [I-D.palombini-ace-key-groupcomm]. The payload (formatted as a CBOR map) MUST contain:

- the following fields from the Authorization Response (Section 3.2 of [I-D.palombini-ace-key-groupcomm]):
  - profile set to "coap_pubsub"
  - scope parameter (optionally), set to a CBOR array containing the broker's topic as first element and the string "publisher"

for publishers and "subscriber" for subscribers as second element

- the following fields from the Key Distribution Response (Section 4.2 of [I-D.palombini-ace-key-groupcomm]):
  - "key" parameter including:
    - kty with value 4 (symmetric)
    - alg with value defined by the AS2 (Content Encryption Algorithm)
    - Base IV with value defined by the AS2
    - k with value the symmetric key value
    - OPTIONALLY, exp with the expiration time of the key
    - OPTIONALLY, kid with an identifier for the key value
  - "pub_keys", containing the public keys of all authorized signing members, if the "get_pub_keys" parameter was present and set to the empty array in the Authorization + Key Distribution Request

Examples for the response payload are detailed in Figure 6 and Figure 9.

4. Publisher

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

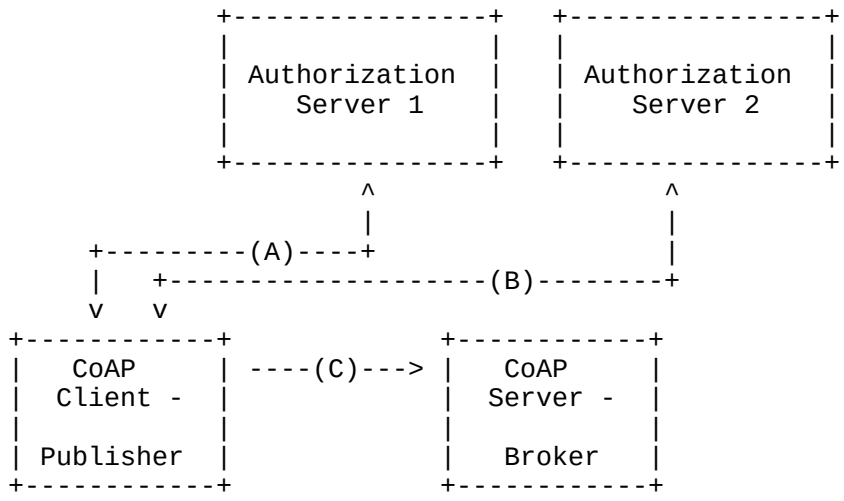


Figure 4: Phase 1: Publisher side

This is a combination of two independent phases:

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

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

In detail:

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

(C) corresponds to the exchange between Publisher and Broker, where the Publisher sends its access token to the Broker and establishes a secure connection with the Broker. Depending on the Information received in (A), this can be for example DTLS handshake, or other protocols. Depending on the application, there may not be the need for this set up phase: for example, if OSCOAP is used directly.

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

(B) corresponds to the retrieval of the keying material to protect the publication, and uses [I-D.palombini-ace-key-groupcomm]. The details are defined in Section 3.1.

An example of the payload of an Authorization + Key Distribution Request and corresponding Response for a Publisher is specified in Figure 5 and Figure 6.

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

Figure 5: Authorization + Key Distribution Request payload for a Publisher

```
    {
      "profile" : "coap_pubsub",
      "key" : {1: 4, 2: h'1234', 3: 12, 5: h'1f389d14d17dc7',
      -1:   h'02e2cc3a9b92855220f255fff1c615bc'}
    }
```

Figure 6: Authorization + Key Distribution Response payload for a Publisher

5. Subscriber

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

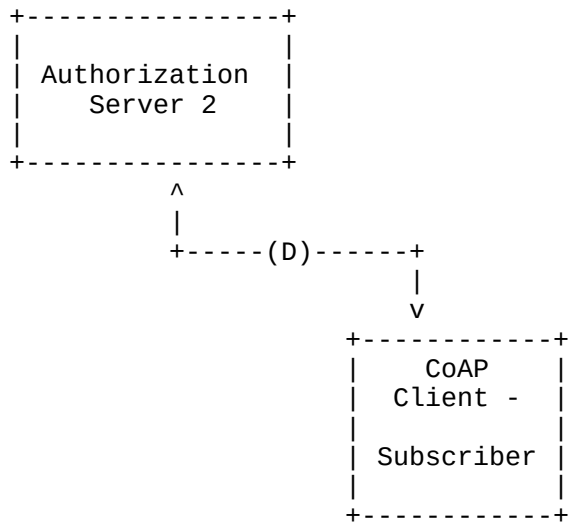


Figure 7: Phase 2: Subscriber side

Step (D) between Subscriber and AS2 corresponds to the retrieval of the keying material to verify the publication. The details are defined in Section 3.1

This step is the same as (B) between Publisher and AS2 (Section 3.1), with the following differences:

- The Authorization + Key Distribution Request MUST NOT contain the client_cred parameter, the role element in the 'scope' parameter MUST be set to "subscriber". The Subscriber MUST have access to the public keys of all the Publishers; this MAY be achieved in the Authorization + Key Distribution Request by using the parameter get_pub_keys set to empty array.

- The Authorization + Key Distribution Response MUST contain the pub_keys parameter.

An example of the payload of an Authorization + Key Distribution Request and corresponding Response for a Subscriber is specified in Figure 8 and Figure 9.

```
               {
                 "grant_type" : "client_credentials",
                 "scope" : ["Broker1/Temp", "subscriber"],
                 "get_pub_keys" : [ ]
               }
```

Figure 8: Authorization + Key Distribution Request payload for a Subscriber

```
{
  "profile" : "coap_pubsub",
  "scope" : ["Broker1/Temp", "subscriber"],
  "key" : {1: 4, 2: h'1234', 3: 12, 5: h'1f389d14d17dc7',
  -1: h'02e2cc3a9b92855220f255fff1c615bc'},
  "pub_keys" : [
   {
      1 : 2, / type EC2 /
      2 : h'11', / kid /
      3 : -7, / alg ECDSA with SHA-256 /
      -1 : 1 , / crv P-256 /
      -2 : h'65eda5a12577c2bae829437fe338701a10aaa375e1bb5b5de108de43
      9c08551d', / x /
      -3 : h'1e52ed75701163f7f9e40ddf9f341b3dc9ba860af7e0ca7ca7e9eecd
      0084d19c' / y /
    }
  ]
}
```

Figure 9: Authorization + Key Distribution Response payload for a Subscriber

6. Pub-Sub Protected Communication

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

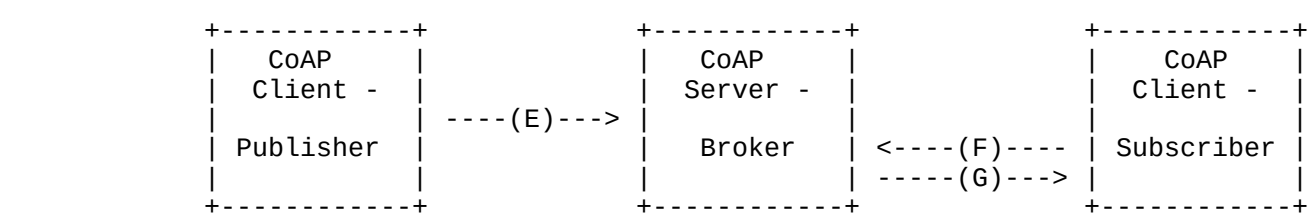


Figure 10: Phase 3: Secure communication between Publisher and Subscriber

The (E) message corresponds to the publication of a topic on the Broker. The publication (the resource representation) is protected with COSE ([RFC8152]). The (F) message is the subscription of the Subscriber, which is unprotected. The (G) message is the response from the Broker, where the publication is protected with COSE.

The flow graph is presented below.

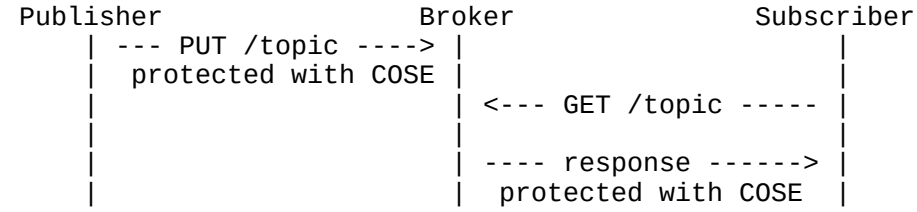


Figure 11: (E), (F), (G): Example of protected communication

6.1. Using COSE Objects to protect the resource representation

The Publisher uses the symmetric COSE Key received from AS2 in exchange B (Section 3.1) to protect the payload of the PUBLISH operation (Section 4.3 of [I-D.ietf-core-coap-pubsub]). Specifically, the COSE Key is used to create a COSE_Encrypt0 with algorithm specified by AS2. The Publisher uses the private key corresponding to the public key sent to the AS2 in exchange B (Section 3.1) to countersign the COSE Object as specified in Section 4.5 of [RFC8152]. The CoAP payload is replaced by the COSE object before the publication is sent to the Broker.

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

The COSE object is constructed in the following way:

- The protected Headers (as described in Section 3 of [RFC8152]) MAY contain the kid parameter, with value the kid of the symmetric COSE Key received in Section 3.1 and MUST contain the content encryption algorithm

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

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

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

* the signature computed as specified in Section 4.5 of [RFC8152]

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

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

An example is given in Figure 12

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

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

The encryption and decryption operations are described in sections 5.3 and 5.4 of [RFC8152].

7. Security Considerations

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

be set up and managed by the same entity having control of the topic, i.e. AS2.

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

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

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

TODO: expand on security and Privacy considerations

8. 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 "[[This document]]" with the RFC number of this specification and delete this paragraph.

Name: coap_pubsub

Description: Profile for delegating client authentication and authorization for publishers and subscribers in a pub-sub setting scenario in a constrained environment.

CBOR Key: TBD

Reference: [[This document]]

9. References

9.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-12 (work in progress), May 2018.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (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>.

[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>.

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-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-03 (work in progress), March 2018.

[I-D.ietf-ace-oscore-profile]
Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-01 (work in progress), March 2018.

Acknowledgments

The author wishes to thank Ari Keraenen, John Mattsson, Ludwig Seitz, Goeran Selander, Jim Schaad and Marco Tiloca for the useful discussion and reviews that helped shape this document.

Author’s Address

Francesca Palombini
Ericsson

Email: francesca.palombini@ericsson.com

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

F. Palombini
Ericsson AB
M. Tiloca
RISE AB

October 22, 2018

Key Provisioning for Group Communication using ACE
draft-palombini-ace-key-groupcomm-02

Abstract

This document defines message formats and procedures for requesting and distributing group keying material using the ACE framework, to protect communications between group members.

Status of This Memo

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

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

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

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

Copyright Notice

Table of Contents

1. Introduction

This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to define the format of messages used to request, distribute and renew the keying material in a group communication scenario, e.g. based on multicast [RFC7390] or on publishing-subscribing [I-D.ietf-core-coap-pubsub].

Profiles that use group communication can build on this document to specify the selection of the message parameters defined in this document to use and their values. Known applications that can benefit from this document would be, for example, profiles addressing group communication based on multicast [RFC7390] or publishing/ subscribing [I-D.ietf-core-coap-pubsub] in ACE.

If the application requires backward and forward security, updated keying material is generated and distributed to the group members (rekeying), when membership changes. A key management scheme performs the actual distribution of the updated keying material to

the group. In particular, the key management scheme rekeys the current group members when a new node joins the group, and the remaining group members when a node leaves the group. This document provides a message format for group rekeying that allows to fulfill these requirements. Rekeying mechanisms can be based on [RFC2093], [RFC2094] and [RFC2627].

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.

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz] and [RFC8152], such as Authorization Server (AS) and Resource Server (RS).

2. Overview

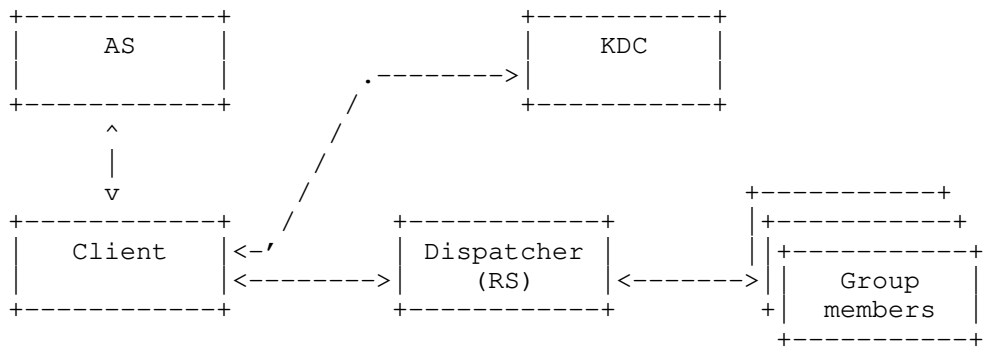


Figure 1: Key Distribution Participants

The following participants (see Figure 1) take part in the authorization and key distribution.

- Client (C): node that wants to join the group communication. It can request write and/or read rights.

- Authorization Server (AS): same as AS in the ACE Framework; it enforces access policies, and knows if a node is allowed to join the group with write and/or read rights.

- Key Distribution Center (KDC): maintains the keying material to protect group communications, and provides it to Clients

authorized to join the group. During the first part of the exchange (Section 3), it takes the role of the RS in the ACE Framework. During the second part (Section 4), which is not based on the ACE Framework, it distributes the keying material. In addition, it provides the latest keying material to group members when requested. If required by the application, the KDC renews and re-distributes the keying material in the group when membership changes.

- Dispatcher: entity through which the Clients communicate with the group and which distributes messages to the group members. Examples of dispatchers are: the Broker node in a pub-sub setting; a relayer node for group communication that delivers group messages as multiple unicast messages to all group members; an implicit entity as in a multicast communication setting, where messages are transmitted to a multicast IP address and delivered on the transport channel.

This document specifies the message flows and formats for:

- Authorizing a new node to join the group (Section 3), and providing it with the group keying material to communicate with the other group members (Section 4).

- Removing of a current member from the group (Section 5).

- Retrieving keying material as a current group member (Section 6 and Section 7).

- Renewing and re-distributing the group keying material (rekeying) upon a membership change in the group (Section 4.2 and Section 5).

Figure 2 provides a high level overview of the message flow for a node joining a group communication setting.

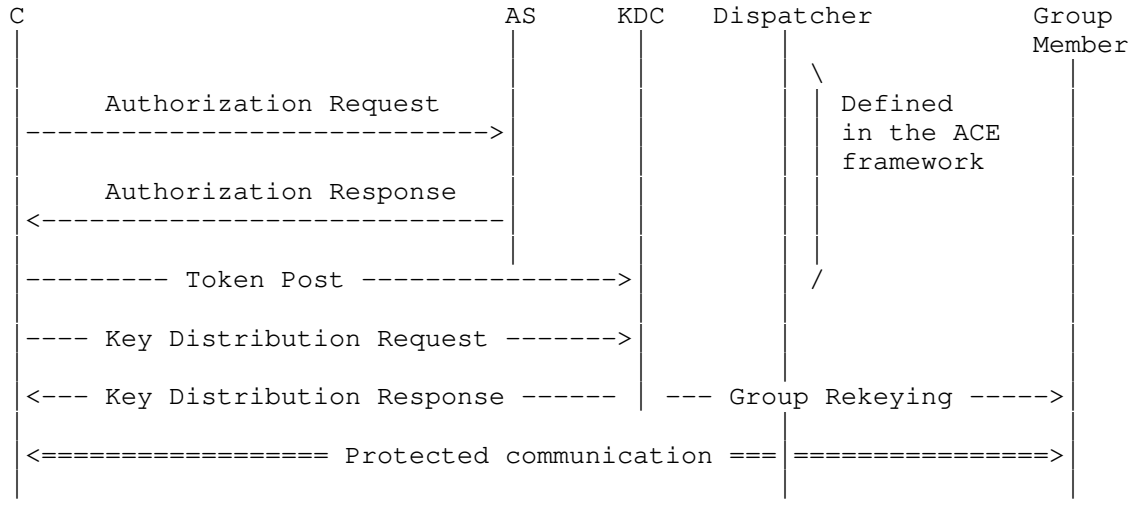


Figure 2: Message Flow Upon New Node's Joining

The exchange of Authorization Request and Authorization Response between Client and AS MUST be secured, as specified by the ACE profile used between Client and KDC.

The exchange of Key Distribution Request and Key Distribution Response between Client and KDC MUST be secured, as a result of the ACE profile used between Client and KDC.

All further communications between the Client and the KDC MUST be secured, for instance with the same security mechanism used for the Key Distribution exchange.

All further communications between a Client and the other group members MUST be secured using the keying material provided in Section 4.

3. Authorization to Join a Group

This section describes in detail the format of messages exchanged by the participants when a node requests access to a group. The first part of the exchange is based on ACE [I-D.ietf-ace-oauth-authz].

As defined in [I-D.ietf-ace-oauth-authz], the Client requests from the AS an authorization to join the group through the KDC (see Section 3.1). If the request is approved and authorization is granted, the AS provides the Client with a proof-of-possession access token and parameters to securely communicate with the KDC (see

Section 3.2). Communications between the Client and the AS MUST be secured, and depends on the profile of ACE used.

Figure 3 gives an overview of the exchange described above.

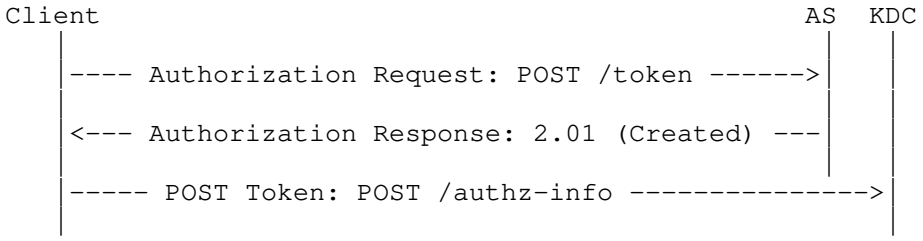


Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

The Authorization Request sent from the Client to the AS is as defined in Section 5.6.1 of [I-D.ietf-ace-oauth-authz] and MUST contain the following parameters:

- 'grant_type', with value "client_credentials".

Additionally, the Authorization Request MAY contain the following parameters, which, if included, MUST have the corresponding values:

- 'scope', with value the identifier of the specific group or topic the Client wishes to access, and optionally the role(s) the Client wishes to take. This value is a CBOR array encoded as a byte string, which contains:

  * As first element, the identifier of the specific group or topic.

  * Optionally, as second element, the role (or CBOR array of roles) the Client wishes to take in the group.

  The encoding of the group or topic identifier and of the role identifiers is application specific.

- 'req_aud', as defined in Section 3.1 of [I-D.ietf-ace-oauth-params], with value an identifier of the KDC.

- 'req_cnf', as defined in Section 3.1 of [I-D.ietf-ace-oauth-params], optionally containing the public key or the certificate of the Client, if it wishes to communicate that to the AS.

- Other additional parameters as defined in [I-D.ietf-ace-oauth-authz], if necessary.

3.2. Authorization Response

The Authorization Response sent from the AS to the Client is as defined in Section 5.6.2 of [I-D.ietf-ace-oauth-authz] and MUST contain the following parameters:

- 'access_token', containing the proof-of-possession access token.

- 'cnf' if symmetric keys are used, not present if asymmetric keys are used. This parameter is defined in Section 3.2 of [I-D.ietf-ace-oauth-params] and contains the symmetric proof-of-possession key that the Client is supposed to use with the KDC.

- 'rs_cnf' if asymmetric keys are used, not present if symmetric keys are used. This parameter is as defined in Section 3.2 of [I-D.ietf-ace-oauth-params] and contains information about the public key of the KDC.

- 'exp', contains the lifetime in seconds of the access token. This parameter MAY be omitted if the application defines how the expiration time is communicated to the Client via other means, or if it establishes a default value.

Additionally, the Authorization Response MAY contain the following parameters, which, if included, MUST have the corresponding values:

- 'scope', which mirrors the 'scope' parameter in the Authorization Request (see Section 3.1). Its value is a CBOR array encoded as a byte string, containing:

  * As first element, the identifier of the specific group or topic the Client is authorized to access.

  * Optionally, as second element, the role (or CBOR array of roles) the Client is authorized to take in the group.

  The encoding of the group or topic identifier and of the role identifiers is application specific.

- Other additional parameters as defined in [I-D.ietf-ace-oauth-authz], if necessary.

The access token MUST contain all the parameters defined above (including the same 'scope' as in this message, if present, or the

'scope' of the Authorization Request otherwise), and additionally other optional parameters the profile requires.

When receiving an Authorization Request from a Client that was previously authorized, and which still owns a valid non expired access token, the AS can simply reply with an Authorization Response including a new access token.

3.3. Token Post

The Client sends a CoAP POST request including the access token to the KDC, as specified in section 5.8.1 of [I-D.ietf-ace-oauth-authz]. If the specific ACE profile defines it, the Client MAY use a different endpoint than /authz-info at the KDC to post the access token to. After successful verification, the Client is authorized to receive the group keying material from the KDC and join the group.

Note that this step could be merged with the following message from the Client to the KDC, namely Key Distribution Request.

4. Key Distribution

This section defines how the keying material used for group communication is distributed from the KDC to the Client, when joining the group as a new member.

If not previously established, the Client and the KDC MUST first establish a pairwise secure communication channel using ACE. The exchange of Key Distribution Request-Response MUST occur over that secure channel. The Client and the KDC MAY use that same secure channel to protect further pairwise communications, that MUST be secured.

During this exchange, the Client sends a request to the AS, specifying the group it wishes to join (see Section 4.1). Then, the KDC verifies the access token and that the Client is authorized to join that group; if so, it provides the Client with the keying material to securely communicate with the member of the group (see Section 4.2).

Figure 4 gives an overview of the exchange described above.

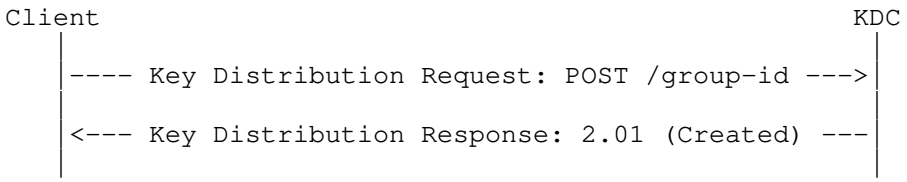


Figure 4: Message Flow of Key Distribution to a New Group Member

The same set of message can also be used for the following cases, when the Client is already a group member:

- The Client wishes to (re-)get the current keying material, for cases such as expiration, loss or suspected mismatch, due to e.g. reboot or missed group rekeying. This is further discussed in Section 6.

- The Client wishes to (re-)get the public keys of other group members, e.g. if it is aware of new nodes joining the group after itself. This is further discussed in Section 7.

Additionally, the format of the payload of the Key Distribution Response (Section 4.2) can be reused for messages sent by the KDC to distribute updated group keying material, in case of a new node joining the group or of a current member leaving the group. The key management scheme used to send such messages could rely on, e.g., multicast in case of a new node joining or unicast in case of a node leaving the group.

Note that proof-of-possession to bind the access token to the Client is performed by using the proof-of-possession key bound to the access token for establishing secure communication between the Client and the KDC.

4.1. Key Distribution Request

The Client sends a Key Distribution request to the KDC. This corresponds to a CoAP POST request to the endpoint in the KDC associated to the group to join. The endpoint in the KDC is associated to the 'scope' value of the Authorization Request/ Response. The payload of this request is a CBOR Map which MAY contain the following fields, which, if included, MUST have the corresponding values:

- 'scope', with value the specific resource that the Client is authorized to access (i.e. group or topic identifier) and role(s), encoded as in Section 3.1.

- 'get_pub_keys', if the Client wishes to receive the public keys of the other nodes in the group from the KDC. The value is an empty CBOR Array. This parameter may be present if the KDC stores the public keys of the nodes in the group and distributes them to the Client; it is useless to have here if the set of public keys of the members of the group is known in another way, e.g. it was provided by the AS.

- 'client_cred', with value the public key or certificate of the Client. If the KDC is managing (collecting from/distributing to the Client) the public keys of the group members, this field contains the public key of the Client.

- 'pub_keys_repos', can be present if a certificate is present in the 'client_cred' field, with value a list of public key repositories storing the certificate of the Client.

4.2. Key Distribution Response

The KDC verifies the access token and, if verification succeeds, sends a Key Distribution success Response to the Client. This corresponds to a 2.01 Created message. The payload of this response is a CBOR Map which MUST contain the following fields:

- 'key', used to send the keying material to the Client, as a COSE_Key ([RFC8152]) containing the following parameters:

  * 'kty', as defined in [RFC8152].

  * 'k', as defined in [RFC8152].

  * 'exp' (optionally), as defined below. This parameter is RECOMMENDED to be included in the COSE_Key. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.

  * 'alg' (optionally), as defined in [RFC8152].

  * 'kid' (optionally), as defined in [RFC8152].

  * 'base iv' (optionally), as defined in [RFC8152].

  * 'clientID' (optionally), as defined in [I-D.ietf-ace-oscore-profile].

  * 'serverID' (optionally), as defined in [I-D.ietf-ace-oscore-profile].

* 'kdf' (optionally), as defined in [I-D.ietf-ace-oscore-profile].

* 'slt' (optionally), as defined in [I-D.ietf-ace-oscore-profile].

* 'cs_alg' (optionally), containing the algorithm value to countersign the message, taken from Table 5 and 6 of [RFC8152].

The parameter 'exp' identifies the expiration time in seconds after which the COSE_Key is not valid anymore for secure communication in the group. A summary of 'exp' can be found in Figure 5.

Name	Label	CBOR Type	Value Registry	Description
exp	TBD	Integer or floating-point number	COSE Key Common Parameters	Expiration time in seconds

Figure 5: COSE Key Common Header Parameter 'exp'

Optionally, the Key Distribution Response MAY contain the following parameters, which, if included, MUST have the corresponding values:

o 'pub_keys', may only be present if 'get_pub_keys' was present in the Key Distribution Request; this parameter is a COSE_KeySet (see [RFC8152]), which contains the public keys of all the members of the group.

o 'group_policies', with value a list of parameters indicating how the group handles specific management aspects. This includes, for instance, approaches to achieve synchronization of sequence numbers among group members. The exact format of this parameter is specific to the profile.

o 'mgt_key_material', with value the administrative keying material to participate in the group rekeying performed by the KDC. The exact format and content depend on the specific rekeying scheme used in the group, which may be specified in the profile.

Specific profiles need to specify how exactly the keying material is used to protect the group communication.

If the application requires backward security, the KDC SHALL generate new group keying material and securely distribute it to all the

current group members, using the message format defined in this section. Application profiles may define alternative message formats.

TBD: define for verification failure

5. Removal of a Node from the Group

This section describes at a high level how a node can be removed from the group.

If the application requires forward security, the KDC SHALL generate new group keying material and securely distribute it to all the current group members but the leaving node, using the message format defined in Section 4.2. Application profiles may define alternative message formats.

5.1. Expired Authorization

If the node is not authorized anymore, the AS can directly communicate that to the KDC. Alternatively, the access token might have expired. If Token introspection is provided by the AS, the KDC can use it as per Section 5.7 of [I-D.ietf-ace-oauth-authz], in order to verify that the access token is still valid.

Either case, once aware that a node is not authorized anymore, the KDC has to remove the unauthorized node from the list of group members, if the KDC keeps track of that.

5.2. Request to Leave the Group

A node can actively request to leave the group. In this case, the Client can send a request formatted as follows to the KDC, to abandon the group.

TBD: Format of the message to leave the group

The KDC should then remove the leaving node from the list of group members, if the KDC keeps track of that.

Note that, after having left the group, a node may wish to join it again. Then, as long as the node is still authorized to join the group, i.e. it has a still valid access token, it can re-request to join the group directly to the KDC without needing to retrieve a new access token from the AS. This means that the KDC needs to keep track of nodes with valid access tokens, before deleting all information about the leaving node.

6. Retrieval of Updated Keying Material

A node stops using the group keying material upon its expiration, according to the 'exp' parameter specified in the retained COSE Key. Then, if it wants to continue participating in the group communication, the node has to request new updated keying material to the KDC.

The Client may perform the same request to the KDC also upon receiving messages from other group members without being able to correctly decrypt them. This may be due to a previous update of the group keying material (rekeying) triggered by the KDC, that the Client was not able to receive or decrypt.

Note that policies can be set up so that the Client sends a request to the KDC only after a given number of unsuccessfully decrypted incoming messages.

6.1. Key Re-Distribution Request

To request a re-distribution of keying material, the Client sends a shortened Key Distribution Request to the KDC (Section 4.1), formatted as follows. The payload MUST contain only the following field:

- 'scope', which contains only the identifier of the specific group or topic, encoded as in Section 3.1. That is, the role field is not present.

6.2. Key Re-Distribution Response

The KDC receiving a Key Re-Distribution Request MUST check that it is storing a valid access token from that client for that scope.

TODO: defines error response if it does not have it / is not valid.

The KDC replies to the Client with a Key Distribution Response containing the 'key' parameter, and optionally 'group_policies' and 'mgt_key_material', as specified in Section 4.2. Note that this response might simply re-provide the same keying material currently owned by the Client, if it has not been renewed.

7. Retrieval of Public Keys for Group Members

In case the KDC maintains the public keys of group members, a node in the group can contact the KDC to request public keys of either all group members or a specified subset, using the messages defined below.

Figure 6 gives an overview of the exchange described above.

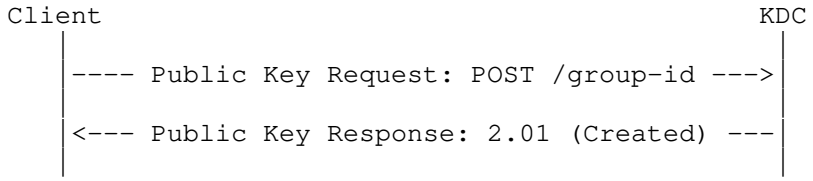


Figure 6: Message Flow of Public Key Request-Response

Note that these messages can be combined with the Key Re-Distribution messages in Section 6, to request at the same time the keying material and the public keys. In this case, either a new endpoint at the KDC may be used, or additional information needs to be sent in the request payload, to distinguish these combined messages from the Public Key messages described below, since they would be identical otherwise.

7.1. Public Key Request

To request public keys, the Client sends a shortened Key Distribution Request to the KDC (Section 4.1), formatted as follows. The payload of this request MUST contain the following fields:

- 'get_pub_keys', which has as value a CBOR array including either:
  - no elements, i.e. an empty array, in order to request the public key of all current group members; or
  - N elements, each of which is the identifier of a group member, in order to request the public key of the specified nodes.
- 'scope', which contains only the identifier of the specific group or topic, encoded as in Section 3.1. That is, the role field is not present.

7.2. Public Key Response

The KDC replies to the Client with a Key Distribution Response containing only the 'pub_keys' parameter, as specified in Section 4.2. The payload of this response contains the following field:

- 'pub_keys', which contains either:

* the public keys of all the members of the group, if the 'get_pub_keys' parameter of the Public Key request was an empty array; or

* the public keys of the group members with the identifiers specified in the 'get_pub_keys' parameter of the Public Key request.

The KDC ignores possible identifiers included in the 'get_pub_keys' parameter of the Public Key request if they are not associated to any current group member.

8. Security Considerations

The KDC must renew the group keying material upon its expiration.

The KDC should renew the keying material upon group membership change, and should provide it to the current group members through the rekeying scheme used in the group.

9. IANA Considerations

The following registration is required for the COSE Key Common Parameter Registry specified in Section 16.5 of [RFC8152]:

- Name: exp
- Label: TBD
- CBOR Type: Integer or floating-point number
- Value Registry: COSE Key Common Parameters
- Description: Identifies the expiration time in seconds of the COSE Key
- Reference: [[this specification]]

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-16 (work in progress), October 2018.

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-00 (work in progress), September 2018.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-04 (work in progress), October 2018.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

10.2. Informative References

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

[RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <https://www.rfc-editor.org/info/rfc2093>.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <https://www.rfc-editor.org/info/rfc2094>.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <https://www.rfc-editor.org/info/rfc2627>.

[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>.

Acknowledgments

The following individuals were helpful in shaping this document: Ben Kaduk, John Mattsson, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

July 02, 2018

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-09

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.

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. 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 uses the Concise Data Definition Language (CDDL) [I-D.ietf-cbor-cddl] to express CBOR data structures [RFC7049]. 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", "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.

2. Protocol Overview

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

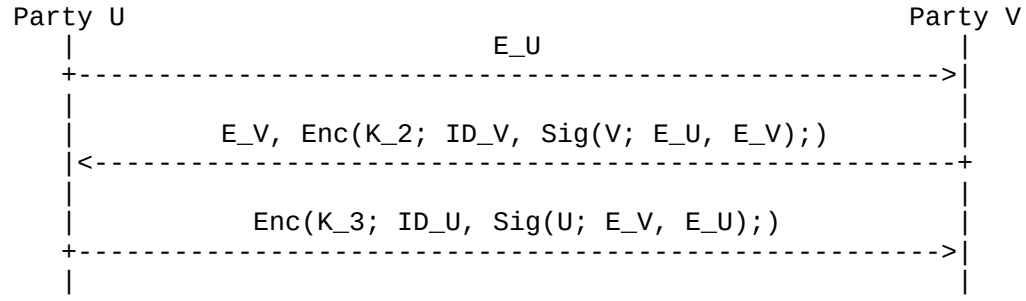


Figure 1: AEAD variant of the SIGMA-I protocol

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

- E_U and E_V are the ECDH ephemeral public keys of U and V, respectively.
- ID_U and ID_V are identifiers for the public keys of U and V, respectively.
- Sig(U; .) and S(V; .) denote signatures made with the private key of U and V, respectively.
- Enc(K; P; A) denotes AEAD encryption of plaintext P and additional authenticated data A using the key K derived from the shared secret. The AEAD MUST NOT be replaced by plain encryption, see Section 8.

As described in Appendix B of [SIGMA], in order to create a "full-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.
- 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.

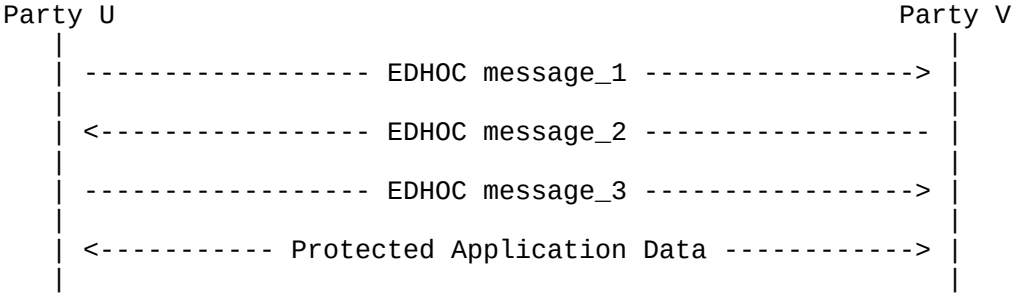


Figure 2: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or certificates. 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 (UAD and PAD) to be sent. Unprotected Application Data (UAD_1, UAD_2) may be sent in message_1 and message_2, while Protected Application Data (PAD_3) may be send in message_3.

3.1. Ephemeral Public Keys

The ECDH ephemeral public keys are formatted as a COSE_Key of type EC2 or OKP according to section 13.1 and 13.2 of [RFC8152], but only a subset of the parameters are included in the EDHOC messages. The curve X25519 is mandatory to implement. For Elliptic Curve Keys of type EC2, compact representation and compact output as per [RFC6090] MAY be used, i.e. the 'y' parameter is 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 and 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 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.

Raw public keys are stored as COSE_Key objects and identified with a 'kid' value, see [RFC8152]. Certificates can be identified in different ways, ID_CRED_U and ID_CRED_V may contain the credential used for authentication (e.g. x5bag or x5chain) or identify the credential used for authentication (e.g. x5t, x5u), see [I-D.schaad-cose-x509]. The full credential (e.g. X.509 certificates or a COSE_Key) are included in CRED_V and CRED_U.

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

EDHOC with asymmetric key authentication is illustrated in Figure 3.

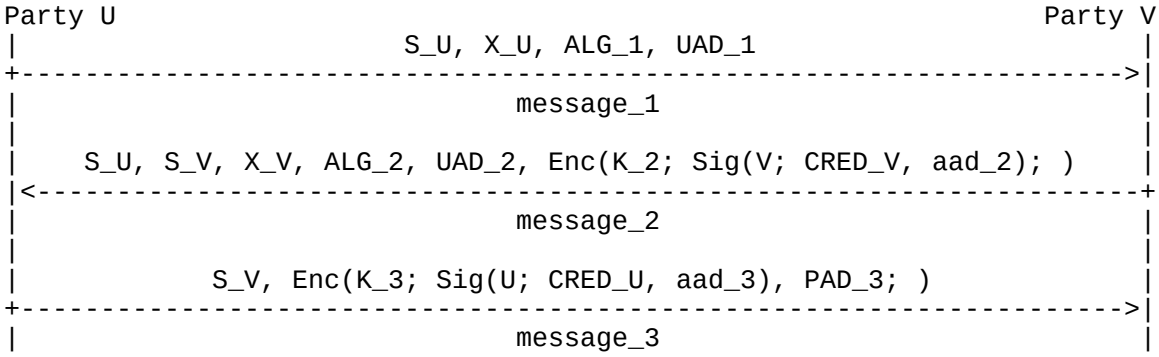


Figure 3: EDHOC with asymmetric key authentication.

4.1.1. Mandatory to Implement Algorithms

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

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  MSG_TYPE : int,
  S_U : bstr,
  ECDH-Curves_U : alg_array,
  ECDH-Curve_U : uint,
  X_U : bstr,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  SIGs_V : alg_array,
  SIGs_U : alg_array,
  ? UAD_1 : bstr
]

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

where:

- MSG_TYPE = 1
- S_U - variable length session identifier
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- ECDH-Curve_U - a single chosen algorithm from ECDH-Curves_U (array index with zero-based indexing)
- X_U - the x-coordinate of ephemeral public key of Party U
- 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
- UAD_1 - bstr containing unprotected 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 generate 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.

- Generate an ephemeral ECDH key pair 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. Let X_U be the x-coordinate of the ephemeral public key.

- Choose a session identifier S_U and store it for the length of the protocol. Party U needs to be able to retrieve the protocol state using the session identifier S_U and other information such as the 5-tuple. 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 that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve indicated by ECDH-Curve_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- Validate that there is a solution to the curve definition for the given x-coordinate X_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. If V does not support the curve ECDH-Curve_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; Z"

where Z is the index of the first curve in ECDH-Curves_U that V supports

- Pass UAD_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,
  CIPHERTEXT_2 : bstr
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr / nil,
  S_V : bstr,
  X_V : bstr,
  HKDF_V : uint,
  AEAD_V : uint,
  SIG_V : uint,
  SIG_U : uint,
)

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
- X_V - the x-coordinate of 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

- H() - the hash function in HKDF_V

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56a] using the curve indicated by ECDH-Curve_U. Format a ephemeral public key as a COSE_key as specified in Section 3.1. Let X_V be the x-coordinate of the ephemeral public key.

- Choose a session identifier S_V and store it for the length of the protocol. Party V needs to be able to retrieve the protocol state using the session identifier S_V and other information such as the 5-tuple. 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.

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

  * COSE_Sign1 = [PROTECTED_2, '', [CRED_V, aad_2], SIGNATURE_2]

  * PROTECTED_2 = { xyz : ID_CRED_V }

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

  * ID_CRED_V - identifier for the public key of Party V, see Section 4.1

  * CRED_V - bstr containing the credential containing the public key of Party V, see Section 4.1

- Compute COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2 and the following parameters.

  * COSE_Encrypt0 = ['', '', CIPHERTEXT_2]

  * plaintext = [PROTECTED_2, SIGNATURE_2, ? UAD_2]

* UAD_2 = bstr containing opaque unprotected application data

o Format message_2 as specified in Section 4.3.1

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

o Retrieve the protocol state using the session identifier S_U and other information such as the 5-tuple.

o Validate that there is a solution to the curve definition for the given x-coordinate X_V.

o Decrypt COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2.

o Verify COSE_Sign1 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.

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,
  CIPHERTEXT_3 : bstr
]

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

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Compute COSE_Sign1 as defined in section 4.4 of [RFC8152], using algorithm SIG_U, the private key of Party U, and the following parameters.
  - COSE_Sign1 = [PROTECTED_3, '', [CRED_U, aad_3], SIGNATURE_3]
  - PROTECTED_3 = { xyz : ID_CRED_U }
  - ID_CRED_U - identifier for the public key of Party U, see Section 4.1
  - CRED_U - bstr containing the credential containing the public key of Party U, see Section 4.1
- Compute COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3 and the following parameters.
  - COSE_Encrypt0 = ['', '', CIPHERTEXT_3]
  - plaintext = [PROTECTED_3, SIGNATURE_3, ? PAD_3]
  - PAD_3 = bstr containing opaque protected application data
- Format message_3 as specified in Section 4.4.1

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Retrieve the protocol state using the session identifier S_V and other information such as the 5-tuple.
- Decrypt COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3.
- Verify COSE_Sign1 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.

o Pass PAD_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 key (PSK) with a good amount of randomness and the requirement that:

o 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.

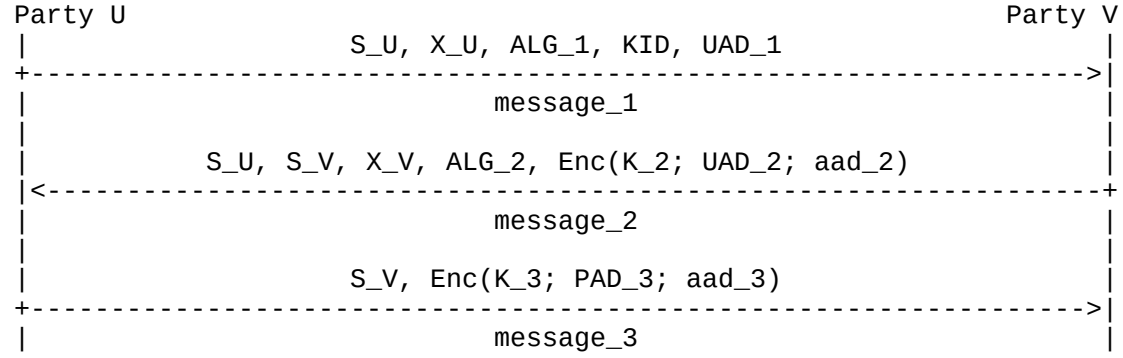


Figure 4: EDHOC with symmetric key authentication.

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,
  ECDH-Curves_U : alg_array,
  ECDH-Curve_U : uint,
  X_U : bstr,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  KID : bstr,
  ? UAD_1 : bstr
)

serialized_COSE_Key = bstr .cbor COSE_Key

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

where:

- MSG_TYPE = 4
- S_U - variable length session identifier
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- ECDH-Curve_U - a single chosen algorithm from ECDH-Curves_U (array index with zero-based indexing)
- X_U - the x-coordinate of ephemeral public key of Party U
- HKDFs_U - supported ECDH-SS w/ HKDF algorithms
- AEADs_U - supported AEAD algorithms
- KID - identifier of the pre-shared key
- UAD_1 - bstr containing unprotected 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 generate 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.

- Generate an ephemeral ECDH key pair 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. Let X_U be the x-coordinate of the ephemeral public key.

- Choose a session identifier S_U and store it for the length of the protocol. Party U needs to be able to retrieve the protocol state using the session identifier S_U and other information such as the 5-tuple. 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 that at least one of each kind of the proposed algorithms are supported.

- Verify that the ECDH curve indicated by ECDH-Curve_U is supported, and that no prior curve in ECDH-Curves_U is supported.

- Validate that there is a solution to the curve definition for the given x-coordinate X_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. If V does not support the curve ECDH-Curve_U, but supports another ECDH curves in ECDH-Curves_U, then the error message MUST include a diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U.

- Pass UAD_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,
  CIPHERTEXT_2 : bstr
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr / nil,
  S_V : bstr,
  X_V : bstr,
  HKDF_V : uint,
  AEAD_V : uint
)

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
- X_V - the x-coordinate of ephemeral public key of Party V
- HKDF_V - an single chosen algorithm from HKDFs_U
- AEAD_V - an single chosen algorithm from AEADs_U
- H() - the hash function in HKDF_V

5.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56a] using the curve indicated by ECDH-Curve_U. Format a ephemeral public key as a COSE_key as specified in Section 3.1. Let X_V be the x-coordinate of the ephemeral public key.

- Choose a session identifier S_V and store it for the length of the protocol. Party V needs to be able to retrieve the protocol state using the session identifier S_V and other information such as the 5-tuple. 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.

- Compute COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_2, and IV_2 and the following parameters.

  * COSE_Encrypt0 = ['', '', CIPHERTEXT_2]

  * external_aad = aad_2

  * plaintext = ? UAD_2

  * UAD_2 = bstr containing opaque unprotected application data

- Format message_2 as specified in Section 5.3.1

5.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Retrieve the protocol state using the session identifier S_U and other information such as the 5-tuple.

- Validate that there is a solution to the curve definition for the given x-coordinate X_V.

- Decrypt and verify COSE_Encrypt0 as 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 UAD_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,
  CIPHERTEXT_3 : bstr
]

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

5.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Compute COSE_Encrypt0 as defined in section 5.3 of [RFC8152], with AEAD_V, K_3, and IV_3 and the following parameters.
  - COSE_Encrypt0 = [’’, ’’, CIPHERTEXT_3]
  - external_aad = aad_3
  - plaintext = ? PAD_3
  - PAD_2 = bstr containing opaque protected application data
- Format message_3 as specified in Section 5.4.1

5.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Retrieve the protocol state using the session identifier S_V and other information such as the 5-tuple.

- Decrypt and verify COSE_Encrypt0 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 PAD_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 UAD_1 and UAD_2 in the asymmetric case, and UAD_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 UAD_1 in the asymmetric case, and UAD_1 and KID in the symmetric case. Party V should be aware that spoofed EDHOC message_1 cannot be detected.

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.

Ephemeral keys MUST NOT be reused, both parties SHALL generate fresh random ephemeral key pairs.

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. References

9.1. 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-x509-02 (work in progress), July 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>.

[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>.

[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>.

[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>.

9.2. 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) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-13 (work in progress), July 2018.

[I-D.ietf-ace-oscore-profile] Seitz, L., Palombini, F., Gunnarsson, M., and G. Selander, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-02 (work in progress), June 2018.

[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-13 (work in progress), June 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-14 (work in progress), July 2018.

[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.

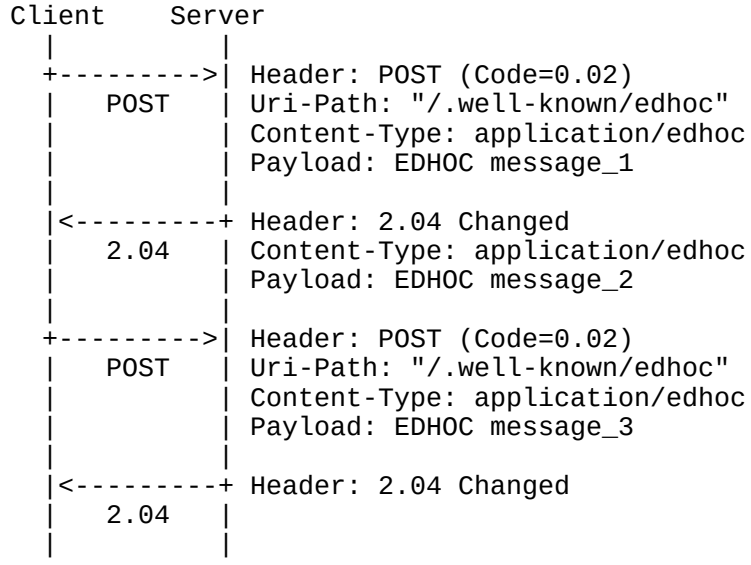


Figure 5: Transferring EDHOC in CoAP

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.

Appendix D. Message Sizes

This appendix gives an estimate of the message sizes when EDHOC is used with Raw Public Keys. Note that the examples in this section and this section are not test vectors, the cryptographic parts are replaces with byte strings of the same length. All examples are given in CBOR diagnostic notation.

```
message1 = [
  1,
  h'c3',
  [4],
  0,
  'abcdefghijklmnopqrstuvwxyz123456',
  [-27],
  [10],
  [-8],
  [-8]
]
```

The size of message_1 is 50 bytes

```
   plaintext = [
     { 4 : 'abba' },
     'abcdefghijklmnopqrstuvwxyz123456abcdefghijklmnopqrstuvwxyz123456'
   ]
```

   The size of plaintext is 74 bytes so the size of ciphertext is 82 bytes

```
message2 = [
  2,
  null,
  h'c4',
  'abcdefghijklmnopqrstuvwxyz123456',
  0,
  0,
  0,
  0,
  'abcdefghijklmnopqrstuvwxyz123456abcdefghijklmnopqrstuvwxyz123456abcdefghijklm
nopqr'
]
```

   The size of message_2 is 127 bytes

```
message3 = [
  3,
  h'c3',
  'abcdefghijklmnopqrstuvwxyz123456abcdefghijklmnopqrstuvwxyz123456abcdefghijklm
nopqr'
]
```

   The size of message_3 is 88 bytes

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.

   We are also grateful to Theis Groenbech Petersen, Thorvald Sahl Joergensen, Alessandro Bruni and Carsten Schuermann for their work on formal analysis of EDHOC.

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: April 25, 2019

M. Tiloca
RISE AB
J. Park
Universitaet Duisburg-Essen
F. Palombini
Ericsson AB

October 22, 2018

Key Management for OSCORE Groups in ACE
draft-tiloca-ace-oscoap-joining-05

Abstract

This document describes a method to request and provision keying material in group communication scenarios where communications are based on CoAP and secured with Object Security for Constrained RESTful Environments (OSCORE). The proposed method delegates the authentication and authorization of new client nodes that join an OSCORE group through a Group Manager server. This approach builds on the ACE framework for Authentication and Authorization, and leverages protocol-specific profiles of ACE to achieve communication security, proof-of-possession and server authentication.

Status of This Memo

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

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

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

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

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 to protect CoAP group communication over IP multicast [RFC7390]. This relies on a Group Manager, which is responsible for managing an OSCORE group, where members exchange CoAP messages secured with OSCORE. The Group Manager can be responsible for multiple groups, coordinates the join process of new group members, and is entrusted with the distribution and renewal of group keying material.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines a method to:

- Authorize a node to join an OSCORE group, and provide it with the group keying material to communicate with other group members.
- Provide updated keying material to group members upon request.
- Renew the group keying material and distribute it to the OSCORE group (rekeying) upon changes in the group membership.

A client node joins an OSCORE group through a resource server acting as Group Manager for that group. The join process relies on an Access Token, which is bound to a proof-of-possession key and authorizes the client to access a specific join resource at the Group Manager.

Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm] for provisioning and renewing keying material in group communication scenarios.

In order to achieve communication security, proof-of-possession and server authentication, the client and the Group Manager leverage protocol-specific profiles of ACE. These include also possible forthcoming profiles that comply with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Readers are expected to be familiar with the terms and concepts related to the CoAP protocol described in [RFC7252][RFC7390]. Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS and /authz-info at the RS. This

document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

Readers are expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE [I-D.ietf-core-object-security] also in group communication scenarios [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications are secured with OSCORE. In this specification, the Group Manager acts as Resource Server.

This document refers also to the following terminology.

- 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.

- Requester: member of an OSCORE group that sends request messages to other members of the group.

- Listener: member of an OSCORE group that receives request messages from other members of the group. A listener may reply back, by sending a response message to the requester which has sent the request message.

- Pure listener: member of a group that is configured as listener and never replies back to requesters after receiving request messages. This corresponds to the term "silent server" used in [I-D.ietf-core-oscore-groupcomm].

- Group rekeying process: the process through which the Group Manager renews the security parameters and group keying material, and (re-)distributes them to the OSCORE group members.

1.2. Relation to Other Documents

Figure 1 overviews the main documents related to this specification. Arrows and asterisk-arrows denote normative references and informative refences, respectively.

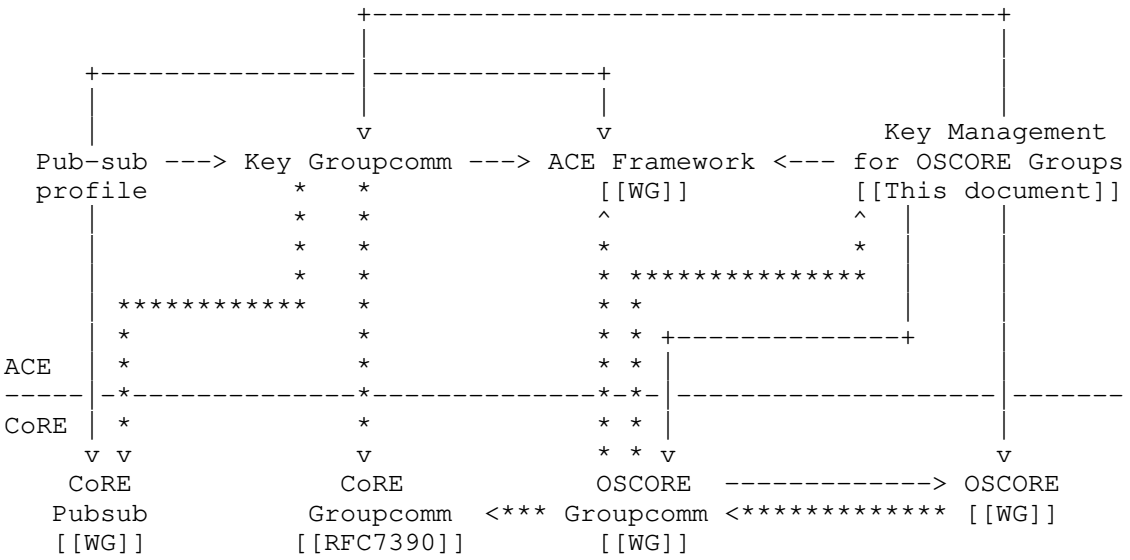


Figure 1: Related Documents

2. Protocol Overview

Group communication for CoAP over IP multicast has been enabled in [RFC7390] and can be secured with Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] as described in [I-D.ietf-core-oscore-groupcomm]. A network node joins an OSCORE group by interacting with the responsible Group Manager. Once registered in the group, the new node can securely exchange messages with other group members.

This specification describes how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to:

- Enable a node to join an OSCORE group through the Group Manager and receive the security parameters and keying material to communicate with the other members of the gorup.

- Enable members of OSCORE groups to retrieve updated group keying material from the Group Manager.

- Enable the Group Manager to renew the security parameters and group keying material, and to (re-)distribute them to the members of the OSCORE group (rekeying).

With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- The Group Manager acts as Resource Server (RS), and hosts one join resource for each OSCORE group it manages. Each join resource is exported by a distinct join endpoint. During the join process, the Group Manager provides joining nodes with the parameters and keying material for taking part to secure communications in the OSCORE group. The Group Manager also maintains the group keying material and performs the group rekeying process to distribute updated keying material to the group members.

- 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) authorizes joining nodes to join OSCORE groups under their respective Group Manager. Multiple Group Managers can be associated to the same AS. The AS MAY release Access Tokens for other purposes than joining OSCORE groups under registered Group Managers. For example, the AS may also release Access Tokens for accessing resources hosted by members of OSCORE groups.

All communications between the involved entities rely on the CoAP protocol and MUST be secured.

In particular, communications between the joining node and the Group Manager leverage protocol-specific profiles of ACE to achieve communication security, proof-of-possession and server authentication. To this end, the AS must signal the specific profile to use, consistently with requirements and assumptions defined in the ACE framework [I-D.ietf-ace-oauth-authz].

With reference to the AS, communications between the joining node and the AS (/token endpoint) as well as between the Group Manager and the AS (/introspect endpoint) can be secured by different means, for instance using DTLS [RFC6347] or OSCORE [I-D.ietf-core-object-security]. Further details on how the AS secures communications (with the joining node and the Group Manager) depend on the specifically used profile of ACE, and are out of the scope of this specification.

2.1. Overview of the Join Process

A node performs the following steps in order to join an OSCORE group. Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm], and are further specified in Section 3 and Section 4 of this document. The Group Manager acts as the Key Distribution Center (KDC) defined in [I-D.palombini-ace-key-groupcomm].

1. The joining node requests an Access Token from the AS, in order to access a join resource on the Group Manager and hence join the associated OSCORE group (see Section 3). The joining node will start or continue using a secure communication channel with the Group Manager, according to the response from the AS.

2. The joining node transfers authentication and authorization information to the Group Manager by posting the obtained Access Token (see Section 4). After that, a joining node must have a secure communication channel established with the Group Manager, before starting to join an OSCORE group under that Group Manager (see Section 4). Possible ways to provide a secure communication channel are DTLS [RFC6347] and OSCORE [I-D.ietf-core-object-security].

3. The joining node starts the join process to become a member of the OSCORE group, by accessing the related join resource hosted by the Group Manager (see Section 4).

4. At the end of the join process, the joining node has received from the Group Manager the parameters and keying material to securely communicate with the other members of the OSCORE group.

5. The joining node and the Group Manager maintain the secure channel, to support possible future communications.

All further communications between the joining node and the Group Manager MUST be secured, for instance with the same secure channel mentioned in step 2.

2.2. Overview of the Group Rekeying Process

If the application requires backward and forward security, the Group Manager MUST generate new security parameters and group keying material, and distribute them to the group (rekeying) upon membership changes.

That is, the group is rekeyed when a node joins the group as a new member, or after a current member leaves the group. By doing so, a

joining node cannot access communications in the group prior its joining, while a leaving node cannot access communications in the group after its leaving.

Parameters and keying material include a new Group Identifier (Gid) for the group and a new Master Secret for the OSCORE Common Security Context of that group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

The Group Manager MUST support the Group Rekeying Process described in Section 7. Future application profiles may define alternative message formats and distribution schemes to perform group rekeying.

3. Joining Node to Authorization Server

This section describes how the joining node interacts with the AS in order to be authorized to join an OSCORE group under a given Group Manager. In particular, it considers a joining node that intends to contact that Group Manager for the first time.

The message exchange between the joining node and the AS consists of the messages Authorization Request and Authorization Response defined in Section 3 of [I-D.palombini-ace-key-groupcomm].

In case the specific AS associated to the Group Manager is unknown to the joining node, the latter can rely on mechanisms like the Unauthorized Resource Request message described in Section 5.1.1 of [I-D.ietf-ace-oauth-authz] to discover the correct AS to contact.

3.1. Authorization Request

The joining node contacts the AS, in order to request an Access Token for accessing the join resource hosted by the Group Manager and associated to the OSCORE group. The Access Token request sent to the /token endpoint follows the format of the Authorization Request message defined in Section 3.1 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The 'scope' parameter MUST be present and MUST include:
  - in the first element, either the Group Identifier (Gid) of the group to join under the Group Manager, or a value from which the Group Manager can derive the Gid of the group to join. It is up to the application to define how the Group Manager possibly performs the derivation of the full Gid. Appendix C of [I-D.ietf-core-oscore-groupcomm] provides an example of structured Gid, composed of a fixed part, namely Group Prefix, and a variable part, namely Group Epoch.

* in the second element, the role(s) that the joining node intends to have in the group it intends to join. Possible values are: "requester"; "listener"; and "pure listener". Possible combinations are: ["requester" , "listener"]; ["requester" , "pure listener"].

- The 'req_aud' parameter MUST be present and is set to the identifier of the Group Manager.

3.2. Authorization Response

The AS is responsible for authorizing the joining node to join specific OSCORE groups, according to join policies enforced on behalf of the respective Group Manager.

In case of successful authorization, the AS releases an Access Token bound to a proof-of-possession key associated to the joining node.

Then, the AS provides the joining node with the Access Token as part of an Access Token response, which follows the format of the Authorization Response message defined in Section 3.2 of [I-D.palombini-ace-key-groupcomm].

The 'exp' parameter MUST be present. Other means for the AS to specify the lifetime of Access Tokens are out of the scope of this specification.

The AS must include the 'scope' parameter in the response when the value included in the Access Token differs from the one specified by the joining node in the request. In such a case, the second element of 'scope' MUST be present and includes the role(s) that the joining node is actually authorized to take in the group, encoded as specified in Section 3.1 of this document.

Also, the 'profile' parameter indicates the specific profile of ACE to use for securing communications between the joining node and the Group Manager (see Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]).

In particular, if symmetric keys are used, the AS generates a proof-of-possession key, binds it to the Access Token, and provides it to the joining node in the 'cnf' parameter of the Access Token response. Instead, if asymmetric keys are used, the joining node provides its own public key to the AS in the 'req_cnf' parameter of the Access Token request. Then, the AS uses it as proof-of-possession key bound to the Access Token, and provides the joining node with the Group Manager's public key in the 'rs_cnf' parameter of the Access Token response.

4. Joining Node to Group Manager

First, the joining node posts the Access Token to the /authz-info endpoint at the Group Manager, in accordance with the Token post defined in Section 3.3 of [I-D.palombini-ace-key-groupcomm]. Then, the joining node establishes a secure channel with the Group Manager, according to what is specified in the Access Token response and to the signalled profile of ACE.

4.1. Join Request

Once a secure communication channel with the Group Manager has been established, the joining node requests to join the OSCORE group, by accessing the related join resource at the Group Manager.

In particular, the joining node sends to the Group Manager a confirmable CoAP request, using the method POST and targeting the join endpoint associated to that group. This join request follows the format and processing of the Key Distribution Request message defined in Section 4.1 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The 'get_pub_keys' parameter is present only if the joining node wants to retrieve the public keys of the group members from the Group Manager during the join process (see Section 6). Otherwise, this parameter MUST NOT be present.

- The 'client_cred' parameter, if present, includes the public key of the joining node. This parameter MAY be omitted if: i) public keys are used as proof-of-possession keys between the joining node and the Group Manager; or ii) the joining node is asking to access the group exclusively as pure listener; or iii) the Group Manager already acquired this information during a previous join process. In any other case, this parameter MUST be present.

4.2. Join Response

The Group Manager processes the request according to [I-D.ietf-ace-oauth-authz]. If this yields a positive outcome, the Group Manager updates the group membership by registering the joining node as a new member of the OSCORE group.

The Group Manager replies to the joining node providing the updated security parameters and keying meterial necessary to participate in the group communication. This join response follows the format and processing of the Key Distribution success Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The 'key' parameter includes what the joining node needs in order to set up the OSCORE Security Context as per Section 2 of [I-D.ietf-core-oscore-groupcomm]. In particular:
  - The 'kty' parameter has value "Symmetric".
  - The 'k' parameter includes the OSCORE Master Secret.
  - The 'exp' parameter specifies when the OSCORE Security Context derived from these parameters expires.
  - The 'alg' parameter, if present, has as value the AEAD algorithm used in the group.
  - The 'kid' parameter, if present, has as value the identifier of the key in the parameter 'k'.
  - The 'base IV' parameter, if present, has as value the OSCORE Common IV.
  - The 'clientID' parameter, if present, has as value the OSCORE Sender ID assigned to the joining node by the Group Manager. This parameter is not present if the node joins the group exclusively as pure listener, according to what specified in the Access Token (see Section 3.2). In any other case, this parameter MUST be present.
  - The 'serverID' parameter MUST be present and has as value the Group Identifier (Gid) associated to the group.
  - The 'kdf' parameter, if present, has as value the KDF algorithm used in the group.
  - The 'slt' parameter, if present, has as value the OSCORE Master Salt.
  - The 'cs_alg' parameter MUST be present and has as value the countersignature algorithm used in the group.
- The 'pub_keys' parameter is present only if the 'get_pub_keys' parameter was present in the join request. If present, this parameter includes the public keys of the group members that are relevant to the joining node. That is, it includes: i) the public keys of the non-pure listeners currently in the group, in case the joining node is configured (also) as requester; and ii) the public keys of the requesters currently in the group, in case the joining node is configured (also) as listener or pure listener.

- The 'group_policies' parameter SHOULD be present and includes a list of parameters indicating particular policies enforced in the group. For instance, it can indicate the method to achieve synchronization of sequence numbers among group members (see Appendix E of [I-D.ietf-core-oscore-groupcomm]).

Finally, the joining node uses the information received in the join response to set up the OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. From then on, the joining node can exchange group messages secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].

If the application requires backward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to all the current group members (see Section 7).

When the OSCORE Master Secret expires, as specified by 'exp' in the 'key' parameter of the join response, the node considers the OSCORE Security Context also invalid and to be renewed. Then, the node retrieves updated security parameters and keying material, by exchanging shortened Join Request and Join Response messages with the Group Manager, according to the approach defined in Section 6 of [I-D.palombini-ace-key-groupcomm]. Finally, the node uses the updated security parameters and keying material to set up the new OSCORE Security Context as described in Section 2 of [I-D.ietf-core-oscore-groupcomm].

5. Leaving of a Group Member

A node may be removed from the OSCORE group, due to expired or revoked authorization, or after its own request to the Group Manager.

If the application requires forward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to the remaining group members (see Section 7). The leaving node must not be able to acquire the new security parameters and group keying material distributed after its leaving.

Same considerations in Section 5 of [I-D.palombini-ace-key-groupcomm] apply here as well, considering the Group Manager acting as KDC. In particular, a node requests to leave the OSCORE group as described in Section 5.2 of [I-D.palombini-ace-key-groupcomm].

6. Public Keys of Joining Nodes

Source authentication of OSCORE messages exchanged within the group is ensured by means of digital counter signatures (see Sections 2 and 3 of [I-D.ietf-core-oscore-groupcomm]). Therefore, group members

must be able to retrieve each other's public key from a trusted key repository, in order to verify source authenticity of incoming group messages.

As also discussed in [I-D.ietf-core-oscore-groupcomm], the Group Manager acts as trusted repository of the public keys of the group members, and provides those public keys to group members if requested to. Upon joining an OSCORE group, a joining node is thus expected to provide its own public key to the Group Manager.

In particular, four cases can occur when a new node joins a group.

- The joining node is going to join the group exclusively as pure listener. That is, it is not going to send messages to the group, and hence to produce signatures with its own private key. In this case, the joining node is not required to provide its own public key to the Group Manager upon joining the group.

- The Group Manager already acquired the public key of the joining node during a previous join process. In this case, the joining node may not provide again its own public key to the Group Manager, in order to limit the size of the join request.

- 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).

Furthermore, as described in Section 4.1, the joining node may have explicitly requested the Group Manager to retrieve the public keys of the current group members, i.e. through the 'get_pub_keys' parameter in the join request. In this case, the Group Manager includes also such public keys in the 'pub_keys' parameter of the join response (see Section 4.2).

Later on as a group member, the node may need to retrieve the public keys of other group members. The node can do that by exchanging shortened Join Request and Join Response messages with the Group Manager, according to the approach defined in Section 7 of [I-D.palombini-ace-key-groupcomm].

7. Group Rekeying Process

In order to rekey the OSCORE group, the Group Manager distributes a new Group ID of the group and a new OSCORE Master Secret for that group. To this end, the Group Manager MUST support at least the following group rekeying scheme. Future application profiles may define alternative message formats and distribution schemes.

The Group Manager uses the same format of the Join Response message in Section 4.2. In particular:

- Only the 'key' parameter is present.
- The 'k' parameter of the 'key' parameter includes the new OSCORE Master Secret.
- The 'serverID' parameter of the 'key' parameter includes the new Group ID.

The Group Manager separately sends a group rekeying message to each group member to be rekeyed. Each rekeying message MUST be secured with the pairwise secure communication channel between the Group Manager and the group member used during the join process.

8. Security Considerations

The method described in this document leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

- Management of group keying material (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager is responsible for the renewal and re-distribution of the keying material in the groups of its competence (rekeying). According to the specific application requirements, this can include rekeying the group upon changes in its membership. In particular, renewing the keying material is required upon a new node's joining or a current node's leaving, in case backward security and forward security have to be preserved, respectively.
- Provisioning and retrieval of public keys (see Section 2 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager acts as key repository of public keys of group members, and provides them upon request.
- Synchronization of sequence numbers (see Section 5 of [I-D.ietf-core-oscore-groupcomm]). This concerns how a listener

node that has just joined an OSCORE group can synchronize with the sequence number of requesters in the same group.

Before sending the join response, the Group Manager should verify that the joining node actually owns the associated private key, for instance by performing a proof-of-possession challenge-response, whose details are out of the scope of this specification.

Further security considerations are inherited from [I-D.palombini-ace-key-groupcomm], the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the specific profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

9. IANA Considerations

This document has no actions for IANA.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-16 (work in progress), October 2018.

[I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-04 (work in progress), October 2018.

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

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Secure group communication for CoAP", draft-ietf-core-oscore-groupcomm-02 (work in progress), June 2018.

[I-D.palombini-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-palombini-ace-key-groupcomm-02 (work in progress), October 2018.

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

10.2. Informative References

[I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-05 (work in progress), October 2018.

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

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

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <https://www.rfc-editor.org/info/rfc7390>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Martin Gunnarsson, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok for their comments and feedback.

The work on this document has been partly supported by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

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

Email: marco.tiloca@ri.se

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com