ACE
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

P. van der Stok
Consultant
P. Kampanakis
Cisco Systems
S. Kumar
Philips Lighting Research
M. Richardson
SSW
M. Furuhed
Nexus Group
S. Raza
RISE SICS
July 2, 2018

# EST over secure CoAP (EST-coaps)
draft-ietf-ace-coap-est-04

## 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 low-resource 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 January 3, 2019.

Table of Contents

## 1. 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 messages run 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 [RFC5246], HTTP [RFC7230] and TCP.

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

This specification 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.

## 2. Terminology

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

Many of the concepts in this document are taken over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

## 3. Conformance to RFC7925 profiles

This section shows 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 request for a trusted certificate list. Private keys can be transported as responses to a

request to a server-side keygeneration as described in section 4.4 of [RFC7030] and discussed in Section 4.5 of this document.

As per [RFC7925] section 3.3 and section 4.4, the mandatory cipher suite for DTLS in EST-coaps is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251], and the curve secp256r1 MUST be supported [RFC4492]; 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 applies.

DTLS1.2 implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]. Uncompressed point format MUST also be supported. [RFC6090] can be used as summary of the ECC algorithms. DTLS 1.3 implementations differ from DTLS 1.2 because they do not support point format negotiation in favor of a single point format for each curve and thus support for DTLS 1.3 does not mandate point formation extensions and negotiation.

The EST-coaps client MUST be configured with at least an implicit TA database from its manufacturer. The authentication of the EST-coaps server by the EST-coaps client is based on certificate authentication in the DTLS handshake.

The authentication of the EST-coaps client is based on 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-installed certificate or a certificate issued by some other party); the server is expected to trust the manufacturer’s root CA certificate in this case.

## 4. Protocol Design

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) fragmentation of UDP datagrams. The use of "Block" for the transfer of larger EST messages is specified in Section 4.6. The Figure 1 below shows the layered EST-coaps architecture.

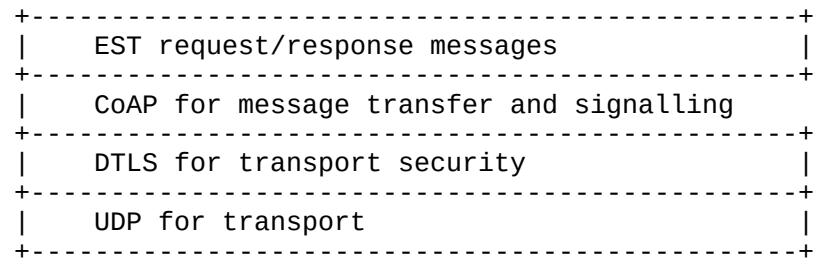



Figure 1: EST-coaps protocol layers

The EST-coaps protocol design follows closely the EST design. The actions supported by EST-coaps are identified by their message types:

- CA certificate retrieval, needed to receive the complete set of CA certificates.
- Simple enroll and reenroll, for CA to sign public client-identity key.
- Certificate Signing Request (CSR) Attributes request messages, informs the client of the fields to include in generated CSR.
- Server-side key generation messages, to provide a private client-identity key when the client choses for an external entity to generate its private key.

## 4.1. Payload format

The content-format (media type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The media types specified in the HTTP Content-Type header (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 used for CoAP MUST map to an allowed combination of URI and media type as defined for EST. The required content-formats for these requests and response messages are defined in Section 9. The CoAP response codes are defined in Section 4.3.

EST-coaps is designed for use between 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. Therefore, the content formats specification in Section 4.1.1 specifies that the binary payload is transported as a CBOR major type 2, a byte string, for all EST-coaps Content-Formats. In the examples of Appendix A, the base16 diagnostic notation is used for CBOR major type 2, where h'450aafbb' represents an example binary payload.

#### 4.1.1. Content Format application/multipart-core

A representation with content format ID TBD8 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.fossati-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 ID of the following representation.

For example, a collection containing two representations, one with content format ID TBD5 and one with content format ID TBD2, looks like this in diagnostic CBOR notation: [TBD5,h'0123456789abcdef',TBD2,h'fedcba9876543210']. An example is shown in Appendix A.4.

### 4.2. Message Bindings

The general EST CoAP 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, and Location-Path in CoAP. These CoAP Options are used to communicate the HTTP fields specified in the EST REST messages.
- EST URLs are HTTPS based (https://), in CoAP these will be assumed to be transformed to coaps (coaps://)

Appendix A includes some practical examples of EST messages translated to CoAP.

### 4.3. CoAP response codes

Section 5.9 of [RFC7252] specifies the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request, 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 POST EST requests. Response code HTTP 202 has no equivalent in CoAP. In Section 4.4 it is specified how EST requests over CoAP handle delayed messages.

All other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

### 4.4. Delayed Responses

Appendix B.2 shows an example of a server response that comes immediately after a client request. The example shows the flows of blocks as the large messages require fragmentation. But 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 a enroll request with an empty ACK with code 0.00, before sending the certificate to the server after a short delay. Consecutively, the server will need more than one "Block2" blocks to respond if the certificate is large. This situation is shown in Figure 2 where a client sends an enrollment request that uses more than one "Block1" blocks. The server uses an empty 0.00 ACK to announce the response which will be provided later with 2.04 messages containing "Block2" options. Having received the first 128 bytes in the first "block2" block, the client asks for a block reduction to 128 bytes in all following "block2" blocks, starting with the second block (NUM=1).

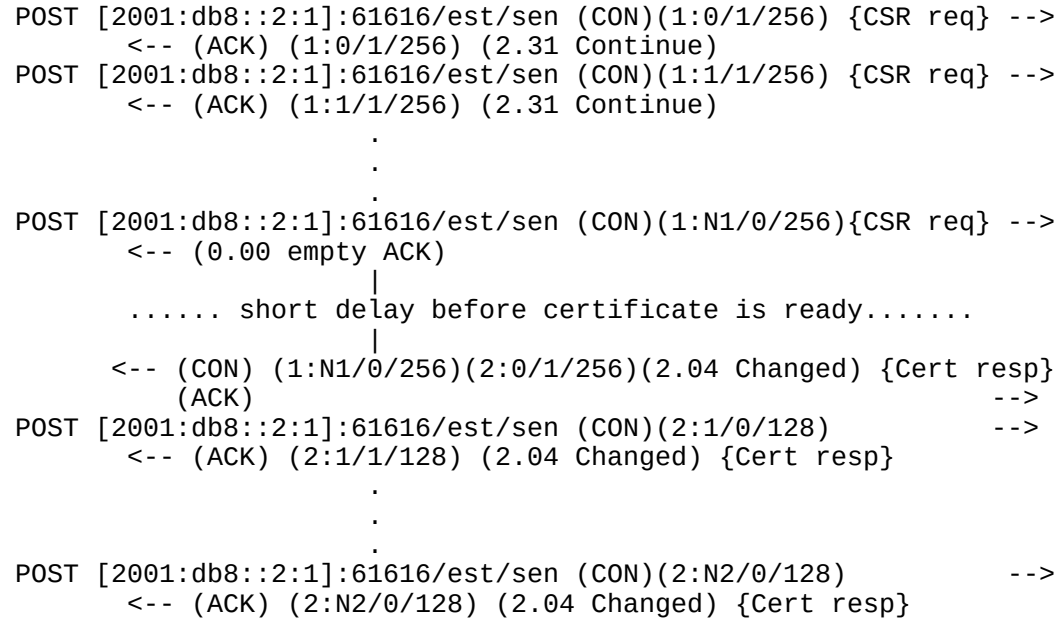



Figure 2: EST-COAP enrolment with short wait

If the server is very slow providing the response (say minutes, possible when a manual intervention is wanted), 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), the client can resend the enrolment request until the server responds with the certificate or the client abandons for other reasons.

To demonstrate this situation, Figure 3 shows a client sending an enrolment request that will use more than one "Block1" block to send the CSR to the server. The server needs more than one "Block2" blocks to respond, but also needs to take a long delay (minutes) to provide the response. Consequently, the server will use a 5.03 ACK for the response. The client can be requested to wait multiple times for a period of Max-Age. Note that in the example below the server asks for a decrease in the block size when acknowledging the first Block2.

Figure 5 can be compared with Figure 3 to see the extra requests after a Max-Age wait.

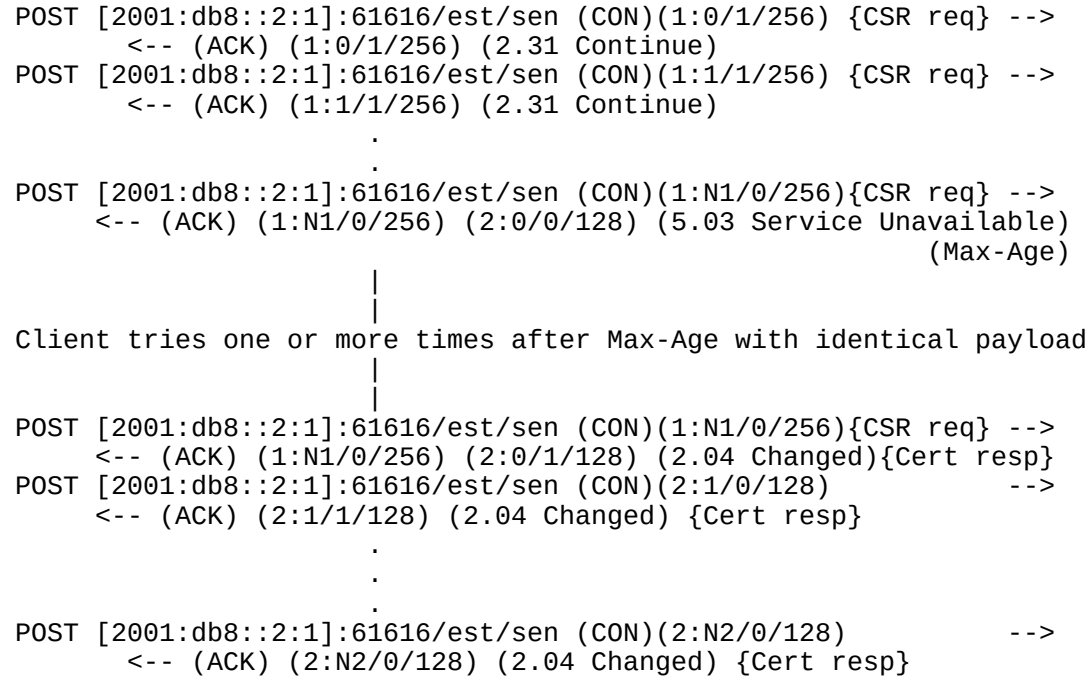



Figure 3: EST-COAP enrolment with long wait

## 4.5. 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 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. Studies have shown that the same keys are generated by the same model devices deployed on-line.

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 is transferred back to the client in the server-side key generation response.

[RFC7030] recommends for the private key returned by the server to be encrypted. The specification provides two methods to encrypt the generated key, symmetric and asymmetric. The methods are signalled by the client by using the relevant attributes (SMIMECapabilities and DecryptKeyIdentifier or AsymmetricDecryptKeyIdentifier) in the CSR request. In the symmetric key case, the key can be established out-of-band or alternatively derived by the established TLS connection as described in [RFC5705].

The sever-side key generation response is returned using a CBOR array Section 4.1.1. The certificate part exactly matches the response from a enrollment response. The private key is placed inside of a CMS SignedData. The SignedData is signed by the party that generated the private key, which may or may not 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].

## 4.6. Message fragmentation

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states that to avoid using IP fragmentation, which involves error-prone datagram reconstitution, invokers of the DTLS record layer SHOULD size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 network SHOULD attempt to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames.

That is not always possible. Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacerts response from the server can include multiple certs that amount to large payloads. Section 4.6 of CoAP [RFC7252] describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Section 4.6 of [RFC7252] also suggests that IPv4 implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes. From [RFC0791] follows that the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Thus, even with ECC certs, EST-coaps messages can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919] as explained in section 2 of [RFC7959]. EST-coaps needs to be able to fragment EST

messages into multiple DTLS datagrams. Fine-grained fragmentation of EST messages is essential.

To perform fragmentation in CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload of a CoAP flow.

The BLOCK draft defines SZX in the Block1 and Block2 option fields. These are used to convey the size of the blocks in the requests or responses.

The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size. As explained in Section 1 of [RFC7959]), blockwise transfers SHOULD be used in Confirmable CoAP messages to avoid the exacerbation of lost blocks.

The Size1 response MAY be parsed by the client as a size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

Examples of fragmented messages are shown in Appendix B.

### 4.7. Deployment limits

Although EST-coaps paves the way for the utilization of EST for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with EST-coaps. The specification of EST-coaps is intended to ensure that EST works for networks of constrained devices that choose to limit their communications stack to UDP/CoAP. It is up to the network designer to decide which devices execute the EST protocol and which do not.

## 5. Discovery and URI

EST-coaps is targeted to low-resource networks with small packets. Saving header space is important and an additional EST-coaps URI is specified that is shorter than the EST URI.

In the context of CoAP, the presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "ace.est" [RFC6690]. Upon success, the return payload will contain

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

The individual EST-coaps server URIs differ from the EST URI by replacing the scheme https by coaps and by specifying shorter resource path names:

coaps://www.example.com/.well-known/est/ArbitraryLabel/<short-est>.

The ArbitraryLabel Path-Segment SHOULD be of the shortest length possible.

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

The short resource URIs MUST be supported. The corresponding longer URIs specified in [RFC7030] MAY be supported.

When discovering the root path for the EST resources, the server MAY return all available resource paths and the used content types. This is useful when multiple content types are specified for EST-coaps server. The example below shows the discovery of the presence and location of management data.

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

  RES: 2.05 Content
</est>; rt="ace.est"
</est/crts>;ct=TBD2
</est/sen>;ct=TBD2 TBD7
</est/sren>;ct=TBD2 TBD7
</est/att>;ct=TBD6
</est/skg>;ct=TBD1 TBD7 TBD8
```

The first line of the discovery response MUST be returned. The five consecutive lines MAY be returned. The return of the content-types in the last four lines allows the client to choose the most appropriate one from multiple content types.

## 6. DTLS Transport Protocol

EST-coaps depends on a secure transport mechanism over UDP that can secure (confidentiality, authenticity) the exchanged CoAP messages.

DTLS is one such secure protocol. When "TLS" is referred to in the context of EST, it is understood that in EST-coaps, security is provided using DTLS instead. No other changes are necessary (all provisional modes etc. are the same as for TLS).

CoAP was designed to avoid fragmentation. DTLS is used to secure CoAP messages. However, fragmentation is still possible at the DTLS layer during the DTLS handshake when using ECC ciphersuites. If fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over several records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

CoAP and DTLS can provide proof of identity for EST-coaps clients and server with simple PKI messages conformant to section 3.1 of [RFC5272]. EST-coaps supports the certificate types and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

Channel-binding information for linking proof-of-identity with connection-based proof-of-possession is optional for EST-coaps. When proof-of-possession is desired, a set of actions are required regarding the use of tls-unique, described in section 3.5 in [RFC7030]. The tls-unique information translates to the contents of the first "Finished" message in the (D)TLS handshake between server and client [RFC5929]. The client is then supposed to add this "Finished" message as a ChallengePassword in the attributes section of the PKCS#10 Request Info to prove that the client is indeed in control of the private key at the time of the TLS session when performing a /simpleenroll, for example. In the case of EST-coaps, the same operations can be performed during the DTLS handshake. 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

```
PRF(master_secret, finished_label, Hash(handshake_messages))
   [0..verify_data_length-1];
```

MUST be computed as if each handshake message had been sent as a single fragment [RFC6347]. Similarly, for DTLS 1.3, the Finished message

```
HMAC(finished_key,
    Transcript-Hash(Handshake Context,
    Certificate*, CertificateVerify*))

    * Only included if present.
```

MUST be computed as if each handshake message had been sent as a single fragment following the algorithm described in 4.4.4 of [I-D.ietf-tls-tls13].

In a constrained CoAP environment, endpoints can't afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. The DTLS connection SHOULD remain open for persistent EST connections. For example, an EST cacerts request that is followed by a simpleenroll request can use the same authenticated DTLS connection. Given that after a successful enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other. In some cases, such as NAT rebinding, keeping the state of a connection is not possible when devices sleep for extended periods of time. In such occasions, [I-D.rescorla-tls-dtls-connection-id] negotiates a connection ID that can eliminate the need for new handshake and its additional cost.

## 7. HTTPS-CoAPS Registrar

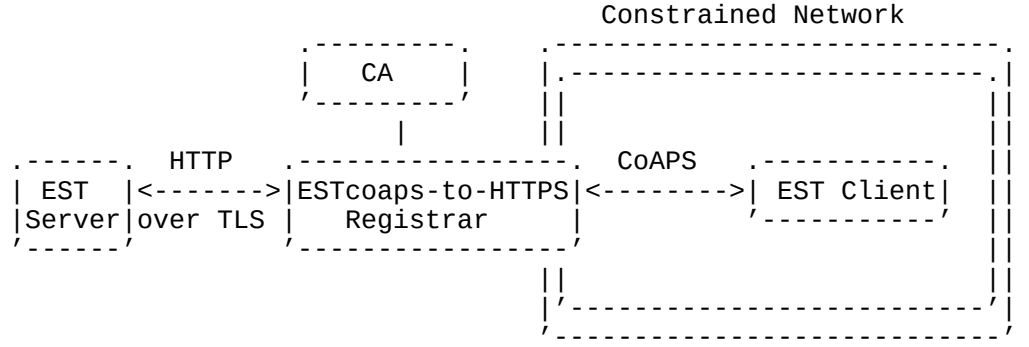
In real-world deployments, the EST server will not always reside within the CoAP boundary. The EST-server can exist outside the constrained network in a non-constrained network that supports TLS/HTTP. In such environments EST-coaps is used by the client within the CoAP boundary and TLS is used to transport the EST messages outside the CoAP boundary. A Registrar at the edge is required to operate between the CoAP environment and the external HTTP network. The EST coaps-to-HTTPS Registrar MUST terminate EST-coaps and authenticate the client downstream and initiate EST connections over TLS upstream.

The Registrar SHOULD authenticate the client downstream and it should be authenticated by the EST server or CA upstream. The Registration Authority (re-)creates the secure connection from DTLS to TLS and vice versa. A trust relationship SHOULD be pre-established between

the Registrar and the EST servers to be able to proxy these connections on behalf of various clients.

When enforcing Proof-of-Possession (POP), the (D)TLS tls-unique value of the (D)TLS session needs to be used to prove that the private key corresponding to the public key is in the possession of and can be used by an end-entity or client. In other words, the CSR the client is using needs to include information from the DTLS connection the client establishes with the server. In EST, that information is the (D)TLS tls-unique value of the (D)TLS session. In the presence of ESTcoaps-to-HTTPS Registrar, the EST-coaps client MUST be authenticated and authorized by the Registrar and the Registrar MUST be authenticated as an EST Registrar client to the EST server. Thus the POP information is lost between the EST-coaps client and the EST server. The EST server becomes aware of the presence of an EST Registrar from its TLS client certificate that includes id-kp-cmcRA [RFC6402] extended key usage extension. 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.

One possible use-case, shown in one figure below, is expected to be deployed in practice:




ESTcoaps-to-HTTPS Registrar at the CoAP boundary.

Table 1 contains the URI mapping between the EST-coaps and EST the Registrar SHOULD adhere to. Section 7 of [RFC8075] and Section 4.3 define the mapping between EST-coaps and HTTP response codes, that determines how the Registrar translates CoAP response codes from/to HTTP status codes. The mapping from Content-Type to media type is defined in Section 9. The conversion from CBOR major type 2 to base64 encoding needs to be done in the Registrar. Conversion is

possible because a TLS link exists between EST-coaps-to-HTTP Registrar and EST server and a corresponding DTLS link exists between EST-coaps-to-HTTP Registrar and EST client.

Due to fragmentation of large messages into blocks, an EST-coaps-to-HTTP Registrar SHOULD reassemble the BLOCKs before translating the binary content to Base-64, 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 of Section 5. The available actions of the Registrars MUST be announced with as many resource paths. The discovery of EST server in the http environment follow the rules specified in [RFC7030].

When server-side key generation is used, if the private key is protected using symmetric keys then the Registrar needs to encrypt the private key down to the client with one symmetric key and decrypt it from the server with another. If no private key encryption takes place the Registrar will be able to see the key as it establishes a separate connection to the server. In the case of asymmetrically encrypted private key, the Registrar may not be able to decrypt it if the server encrypted it with a public key that corresponds to a private key that belongs to the client.

## 8. Parameters

THis section addresses transmission parameters described in sections 4.7 and 4.8 of the CoAP document [RFC7252].

| | |
|---|---|
| ACK_TIMEOUT | 2 seconds |
| ACK_RANDOM_FACTOR | 1.5 |
| MAX_RETRANSMIT | 4 |
| NSTART | 1 |
| DEFAULT_LEISURE | 5 seconds |
| PROBING_RATE | 1 byte/second |

Figure 4: EST-COAP protocol parameters

EST does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting EST. For example, the processing delay of CAs could be less then 2s, but in this case they should send a CoAP ACK every 2s while processing.

The main recommendation, based on experiments using Nexus Certificate Manager with Californium for CoAP support, communicating with a

ContikiOS and tinyDTLS based client, from RISE SICS, is to start with the default CoAP configuration parameters.

However, depending on the implementation scenario, resending and timeouts can also occur on other networking layers, governed by other configuration parameters.

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

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

- NSTART: Limit the number of simultaneous outstanding interactions that a client maintains to a given server. The default is one, hence is the risk of congestion or out-of-order messages already limited.

- 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 the PROBING_RATE setting is not applicable.

Finally, the Table 3 parameters are mainly derived from the more basic Table 2 parameters. If the CoAP implementation allows setting them directly, they might need to be updated if the table 2 parameters are changed.

## 9. IANA Considerations

### 9.1. Content-Format Registry

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are specified in Table 2. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

| Media type | Encoding | ID | Reference |
|---|---|---|---|
| application/pkcs7-mime; smime-type=server-generated-key | - | TBD 1 | [RFC5751] [RFC7030] |
| application/pkcs7-mime; smime-type=certs-only | - | TBD 2 | [RFC5751] |
| application/pkcs7-mime; smime-type=CMC-request | - | TBD 3 | [RFC5751] [RFC5273] |
| application/pkcs7-mime; smime-type=CMC-response | - | TBD 4 | [RFC5751] [RFC5273] |
| application/pkcs8 | - | TBD 5 | [RFC5751] [RFC5958] |
| application/csrattrs | - | TBD 6 | [RFC7030] [RFC7231] |
| application/pkcs10 | - | TBD 7 | [RFC5751] [RFC5967] |
| application/multipart-core | - | TBD 8 | [I-D.fossati-core-multipart-ct] |

Table 2: New CoAP Content-Formats

## 9.2. Resource Type registry

Additions to the sub-registry "CoAP Resource Type", within the "CoRE Parameters" registry are needed for a new resource type.

o rt="ace.est" needs registration with IANA.

# 10. Security Considerations

## 10.1. 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 to encrypt its certificate. The transport of these keys is inherently risky. A full probability analysis MUST be done to establish whether server side key generation enhances or decreases the probability of identity stealing.

When a client uses the Implicit TA database for certificate validation, the client cannot verify that the implicit database can act as an RA. It is RECOMMENDED that such clients include "Linking Identity and POP Information" Section 6 in requests (to prevent such requests from being forwarded to a real EST server by a man in the middle). It is RECOMMENDED that the Implicit Trust Anchor database used for EST server authentication be carefully managed to reduce the chance of a third-party CA with poor certification practices from being trusted. Disabling the Implicit Trust Anchor database after successfully receiving the Distribution of CA certificates response (Section 4.1.3 of [RFC7030]) limits any risk to the first DTLS exchange.

In accordance with [RFC7030], TLS cipher suites that include "_EXPORT_" and "_DES_" in their names MUST NOT be used. More information about recommendations of TLS and DTLS are included in [RFC7525].

As described in CMC, Section 6.7 of [RFC5272], "For keys that can be used as signature keys, signing the certification request with the private key serves as a POP on that key pair". The inclusion of tls-unique in the certification request links the proof-of-possession to the TLS proof-of-identity. This implies but does not prove that the authenticated client currently has access to the private key.

Regarding the Certificate Signing Request (CSR), an adversary could exclude attributes that a server may want, include attributes that a server may not want, and render meaningless other attributes that a server may want. The CA is expected to be able to enforce policies to recover from improper CSR requests.

Interpreters of ASN.1 structures should be aware of the use of invalid ASN.1 length fields and should take appropriate measures to guard against buffer overflows, stack overruns in particular, and malicious content in general.

### 10.2. 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 is used the Registrar terminating the TLS connection establishes a new one with the upstream CA. Thus, it is impossible for POP to be enforced throughout 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.

In a server-side key generation case, depending on the private key encryption method, the Registrar may be able see the private key as it acts as a man-in-the-middle. Thus, the clients puts its trust on the Registrar not exposing the private key.

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 the random number generation using proper entropy. Since the client has no knowledge if the Registrar will be generating the keys and enrolling the certificates with the CA or if the CA will be responsible for generating the keys, the existence of a Registrar requires the client to put its trust on the registrar doing the right thing if it is generating they private keys.

## 11. 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, and John Manuel.

## 12. Change Log

-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 4.5 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

## 13. References

### 13.1. Normative References

[I-D.fossati-core-multipart-ct]
Bormann, C., "Multipart Content-Format for CoAP", draft-fossati-core-multipart-ct-05 (work in progress), June 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (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>.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/info/rfc5272>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <https://www.rfc-editor.org/info/rfc5967>.

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

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

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[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>.

### 13.2. Informative References

[I-D.rescorla-tls-dtls-connection-id]
Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "The Datagram Transport Layer Security (DTLS) Connection Identifier", draft-rescorla-tls-dtls-connection-id-02 (work in progress), November 2017.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <https://www.rfc-editor.org/info/rfc4492>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <https://www.rfc-editor.org/info/rfc4919>.

[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>.

[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>.

[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>.

## Appendix A. EST messages to EST-coaps

This section takes all examples from Appendix A of [RFC7030], changes the payload from Base64 to binary and replaces the http headers by their CoAP equivalents.

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

Binary payload is a CBOR major type 2 (byte array), that is shown with a base16 (hexadecimal) CBOR diagnostic notation.

[EDNOTE: The payloads of the examples need to be re-generated with appropriate tools and example certificates.]

### A.1. cacerts

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

In EST-coaps, a coaps cacerts IPv4 message can be:

GET coaps://192.0.2.1:8085/est/crts

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

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.01 is GET)
  Token = 0x9a (client generated)
  Options
   Option1 (Uri-Host)               [optional]
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)               [optional]
     Option Delta = 0x4  (option nr = 3+4=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4   (option nr = 7+4= 11)
     Option Length = 0x5
     Option Value = "est"
   Option4 (Uri-Path)
     Option Delta = 0x0   (option nr = 11+0= 11)
     Option Length = 0x6
     Option Value = "crts"
   Option5 (Max-Age)
     Option Delta = 0x3   (option nr = 11+3= 14)
     Option Length = 0x1
     Option Value = 0x1    (1 minute)
  Payload = [Empty]
```

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

2.05 Content (Content-Format: TBD2)
{payload}

with CoAP fields

```
     Ver = 1
     T = 2 (ACK)
     Code = 0x45 (2.05 Content)
     Token = 0x9a   (copied by server)
     Options
       Option1 (Content-Format)
         Option Delta = 0xC  (option nr =12)
         Option Length = 0x2
         Option Value = TBD2 (defined in this document)

     Payload =
   h'30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
   c0c3020bb302063c20102020900a61e75193b7acc0d06092a620673410105050030
   1b31193017060355040313106573744578616d706c654341204f774f301e170d313
   330353039033353333315a170d313430353039033353333315a301b3119301706
   03550403131065737445786 16d706c654341204f774f302062300d06092a6206734
   10101050003204f0030204a022041003a923a2968bae4aae136ca4e2512c5200680
   358482ac39d6f640e4574e654ea35f48b1e054c5da3372872f7a1e429f4edf39584
   32efb2106591d3eb783c1034709f251fc86566bda2d541c792389eac4ec9e181f4b
   9f596e5ef2679cc321542b11337f90a44df3c85f1516561fa968a1914f265bc0b82
   76ebe3106a790d97d34c8c37c74fe1c30b396424664ac426284a9f6022e02693843
   6880adfcd95c98ca1dfc2e6d75319b85d0458de28a9d13fb16d620fff7541f6a25d
   7daf00435502030100013 0b040300f0603551d130101f10530030101fc1d0603551
   d0e04160414084d321ca0135e77217a486b686b334b00e0603551d0f0101f104030
   20106300d06092a6206734101050500032041002 3703b965746a0c2c978666d787a
   94f89b495a11f0d369b28936ec2475c0f0855c8e83f823f2b871a1d92282f323c45
   904ba008579216cf5223b8b1bc425a0677262047f7700240631c17f3035d1c3780b
   2385241cba1f4a6e98e6be6820306b3a786de5a557795d1893822347b5f825d34a7
   ad2876f8feba4d525b31066f6505796f71530003431a3e6bbfe788b4565029a7e20
   a51107677552586152d051e8eebf383e92288983421d5c5652a4870c3af74b9bdbe
   d6b462e2263d30f6d3020c330206bc20102020101300d06092a6206734101050500
   301b31193017060355040313106573744578616d706c654341204f774f301e170d3
   133303530393033353333325a170d313430353039033353333325a301b31193017
   06035504031310657374457861 6d706c654341204e774f302062300d06092a62067
   3410101050003204f0030204a02204100ef6b677a3247c1fc03d2b9baf113e5e7e1
   1f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a7229283a790
   8751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b7bd94338
   d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562c4f5abb7
   b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c768d03b8
   076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c14476c37de0f
   55033f192a5ad21f9a2a71c20301000134b050300e0603551d0f0101f104030204c
   1d0603551d0e04160414112966e304761732fbfe6a2c823c301f0603551d2304183
   0165084d321ca0135e77217a486b686b334b00d06092a6206734101050500032041
   00b382ba3355a50e287bae15758b3beff63d34d3e357b90031495d018868e49589b
   9faf46a4ad49b1d35b06ef380106677440934663c2cc111c183655f4dc41c0b3401
   123d35387389db91f1e1b4131b16c291d35730b3f9b33c7475124851555fe5fc647
   e8fd029605367c7e01281bf6617110021b0d10847dce0e9f0ca6c764b6334784055
   172c3983d1e3a3a82301a54fcc9b0670c543a1c747164619101ff23b240b2a26394
```

```
   c1f7d38d0e2f4747928ece5c34627a075a8b3122011e9d9158055c28f020c330206
   bc20102020102300d06092a620673410105050030 1b3119301706035504031310657
   73744578616d706c654341204e774e301e170d313330353039303353333325a170
   d3134303530393033353333325a301b3119301706035504031310657 3744578616d
   706c654341204f774e302062300d06092a62067341010105000 3204f0030204a022
   041003a923a2968bae4aae136ca4e2512c5200680358482ac39d6f640e4574e654e
   a35f48b1e054c5da3372872f7a1e429f4edf3958432efb2106591d3eb783c103470
   9f251fc86566bda2d541c792389eac4ec9e181f4b9f596e5ef2679cc321542b1133
   7f90a44df3c85f1516561fa968a1914f265bc0b8276ebe3106a790d97d34c8c37c7
   4fe1c30b396424664ac426284a9f6022e026938436880adfcd95c98ca1dfc2e6d75
   319b85d0458de28a9d13fb16d620fff7541f6a25d7daf00435502030 1000134b050
   300e0603551d0f0101f104030204c1d0603551d0e04160414084d321ca0135e7721
   7a486b686b334b01f0603551d230418301653112966e304761732fbfe6a2c823c30
   0d06092a620673410105050003204 1002e106933a443070acf5594a3a584d08af7e
   06c295059370a06639eff9bd418d13bc25a298223164a6cf1856b11a81617282e4a
   410d82ef086839c6e2356903227630654553 51e4c596acc7c016b225dec094706c2
   a10608f403b10821984c7c152343b18a768c2ad30238dc45dd653ee6092b0d5cd4c
   2f7d236043269357f76d13f95fb5f00d0e19263c6833948e1ba612ce8197af650e2
   5d882c12f4b6b9b67252c608ef064aca3f9bc867d71172349d510bb7651cd438837
   73d927deb41c4673020bb302063c201020209009b9dda3324700d06092a62067341
   01050500301b3119301706035504031310657 3744578616d706c654341204e774e3
   01e170d313330353039303353333325a170d3134303530393033353333325a301b
   3119301706035504031310657 3744578616d706c654341204e774e302062300d060
   92a620673410101050003204f0030204a02204100ef6b677a3247c1fc03d2b9baf1
   13e5e7e11f49e0421120e6b8384160f2bf02630ef544d5fd0d5623b35713c79a722
   9283a7908751a634aa420a3e2a4b1f10519d046f02f5a5dd6d760c2a842356e067b
   7bd94338d1faa3b3ddd4813060a207b0a097067007e45b052b60fdbae4656e11562
   c4f5abb7b0cf87a79d221f1127313c53371ce1245d63db45a1203a23340ba08042c
   768d03b8076a028d3a51d87d2ef107bbd6f2305ce5e67668724002fb726df9c1447
   6c37de0f55033f192a5ad21f9a2a71c20301000130b040300f0603551d130101f10
   530030101fc1d0603551d0e04160414112966e304761732fbfe6a2c823c300e0603
   551d0f0101f10403020106300d06092a620673410105050003204100423f06d4b76
   0f4b42744a279035571696f272a0060f1325a40898509601ad14004f652db6312a1
   475c4d7cd50f4b269035585d7856c5337765a66b38462d5bdaa7778aab24bbe2815
   e37722cd10e7166c50e75ab75a1271324460211991e7445a2960f47351a1a629253
   34119794b90e320bc730d6c1bee496e7ac125ce9a1eca595a3a4c54a865e6b623c9
   247bfd0a7c19b56077392555c955e233642bec643ae37c166c5e221d797aea3748f
   0391c8d692a5cf9bb71f6d0e37984d6fa673a30d0c006343116f58403100'
```

The hexadecimal dump of the CBOR payload looks like:

```
   59 09CD                                         # bytes(2509)
      30233906092A6206734107028C2A3023260201013100300B06092A62067341070
   18C0C3020BB302063C20102020900A61E75193B7ACC0D06092A62067341010505
   00301B3119301706035504031310657 3744578616D706C654341204F774F301E1
   70D313330353039303353333315A170D3134303530393033353333315A301B31
   1930170603550403131065 73744578616D706C654341204F774F302062300D060
```

92A62067341010105000320 4F0030204A022041003A923A2968BAE4AAE136CA4E
2512C5200680358482AC39D6F640E4574E654EA35F48B1E054C5DA3372872F7A1
E429F4EDF3958432EFB2106591D3EB783C1034709F251FC86566BDA2D541C7923
89EAC4EC9E181F4B9F596E5EF2679CC321542B11337F90A44DF3C85F1516561FA
968A1914F265BC0B8276EBE3106A790D97D34C8C37C74FE1C30B396424664AC42
6284A9F6022E026938436880ADFCD95C98CA1DFC2E6D75319B85D0458DE28A9D1
3FB16D620FFF7541F6A25D7DAF00435502030100013 0B040300F0603551D13010
1F10530030101FC1D0603551D0E04160414084D321CA0135E77217A486B686B33
4B00E0603551D0F0101F10403020106300D06092A62067341010505000320410 0
23703B965746A0C2C978666D787A94F89B495A11F0D369B28936EC2475C0F0855
C8E83F823F2B871A1D92282F323C45904BA008579216CF5223B8B1BC425A06772
62047F7700240631C17F3035D1C3780B2385241CBA1F4A6E98E6BE6820306B3A7
86DE5A557795D1893822347B5F825D34A7AD2876F8FEBA4D525B31066F6505796
F71530003431A3E6BBFE788B4565029A7E20A51107677552586152D051E8EEBF3
83E92288983421D5C5652A4870C3AF74B9BDBED6B462E2263D30F6D3020C33020
6BC20102020101300D06092A62067341010505003 01B31193017060355040313106573744
578616D706C654341204F774F301E170D313330353039303335333332
5A170D3134303530393033353333325A301B311930170603550403131065737445786
16D706C654341204E774F302062300D06092A62067341010105000320 4F00
30204A02204100EF6B677A3247C1FC03D2B9BAF113E5E7E11F49E0421120E6B83
84160F2BF02630EF544D5FD0D5623B35713C79A7229283A7908751A634AA420A3
E2A4B1F10519D046F02F5A5DD6D760C2A842356E067B7BD94338D1FAA3B3DDD48
13060A207B0A097067007E45B052B60FDBAE4656E11562C4F5ABB7B0CF87A79D2
21F1127313C53371CE1245D63DB45A1203A23340BA08042C768D03B8076A028D3
A51D87D2EF107BBD6F2305CE5E67668724002FB726DF9C14476C37DE0F55033F1
92A5AD21F9A2A71C20301000134B050300E0603551D0F0101F104030204C1D060
3551D0E04160414112966E304761732FBFE6A2C823C301F0603551D2304183016
5084D321CA0135E77217A486B686B334B00D06092A62067341010505000320410
0B382BA3355A50E287BAE15758B3BEFF63D34D3E357B90031495D018868E49589
B9FAF46A4AD49B1D35B06EF380106677440934663C2CC111C183655F4DC41C0B3
401123D35387389DB91F1E1B4131B16C291D35730B3F9B33C7475124851555FE5
FC647E8FD029605367C7E01281BF6617110021B0D10847DCE0E9F0CA6C764B633
4784055172C3983D1E3A3A82301A54FCC9B0670C543A1C747164619101FF23B24
0B2A26394C1F7D38D0E2F4747928ECE5C34627A075A8B3122011E9D9158055C28
F020C330206BC20102020102300D06092A62067341010505003 01B31193017060
3550403131065737445786 16D706C654341204E774E301E170D31333035303930
33353333325A170D3134303530393033353333325A301B311930170603550403131
3106573744578616D706C654341204F774E302062300D06092A62067341010105
0003204F0030204A022041003A923A2968BAE4AAE136CA4E2512C520068035848
2AC39D6F640E4574E654EA35F48B1E054C5DA3372872F7A1E429F4EDF3958432E
FB2106591D3EB783C1034709F251FC86566BDA2D541C792389EAC4EC9E181F4B9
F596E5EF2679CC321542B11337F90A44DF3C85F1516561FA968A1914F265BC0B8
276EBE3106A790D97D34C8C37C74FE1C30B396424664AC426284A9F6022E02693
8436880ADFCD95C98CA1DFC2E6D75319B85D0458DE28A9D13FB16D620FFF7541F
6A25D7DAF00435502030100013 4B050300E0603551D0F0101F104030204C1D060
3551D0E04160414084D321CA0135E77217A486B686B334B01F0603551D2304183
01653112966E304761732FBFE6A2C823C300D06092A62067341010505000320 41
002E106933A443070ACF5594A3A584D08AF7E06C295059370A06639EFF9BD418D

```
13BC25A298223164A6CF1856B11A81617282E4A410D82EF086839C6E235690322
763065455351E4C596ACC7C016B225DEC094706C2A10608F403B10821984C7C15
2343B18A768C2AD30238DC45DD653EE6092B0D5CD4C2F7D236043269357F76D13
F95FB5F00D0E19263C6833948E1BA612CE8197AF650E25D882C12F4B6B9B67252
C608EF064ACA3F9BC867D71172349D510BB7651CD43883773D927DEB41C467302
0BB302063C201020209009B9DDA3324700D06092A620673410105050030 1B3119
3017060355040313106573744578616D706C654341204E774E301E170D3133303
5303930333533333325A170D313430353039303335333333 25A301B311930170603
55040313106573744578616D706C654341204E774E302062300D06092A6206734
10101050003204F0030204A02204100EF6B677A3247C1FC03D2B9BAF113E5E7E1
1F49E0421120E6B8384160F2BF02630EF544D5FD0D5623B35713C79A7229283A7
908751A634AA420A3E2A4B1F10519D046F02F5A5DD6D760C2A842356E067B7BD9
4338D1FAA3B3DDD4813060A207B0A097067007E45B052B60FDBAE4656E11562C4
F5ABB7B0CF87A79D221F1127313C53371CE1245D63DB45A1203A23340BA08042C
768D03B8076A028D3A51D87D2EF107BBD6F2305CE5E67668724002FB726DF9C14
476C37DE0F55033F192A5AD21F9A2A71C20301000130B040300F0603551D13010
1F10530030101FC1D0603551D0E04160414112966E304761732FBFE6A2C823C30
0E0603551D0F0101F10403020106300D06092A62067341010505000 3204100423
F06D4B760F4B42744A279035571696F272A0060F1325A40898509601AD14004F6
52DB6312A1475C4D7CD50F4B269035585D7856C5337765A66B38462D5BDAA7778
AAB24BBE2815E37722CD10E7166C50E75AB75A1271324460211991E7445A2960F
47351A1A62925334119794B90E320BC730D6C1BEE496E7AC125CE9A1ECA595A3A
4C54A865E6B623C9247BFD0A7C19B56077392555C955E233642BEC643AE37C166
C5E221D797AEA3748F0391C8D692A5CF9BB71F6D0E37984D6FA673A30D0C00634
3116F58403100
```

### A.2. csrattrs

In the following valid /csrattrs exchange, the EST-coaps client authenticates itself with a certificate issued by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The IPv6 CoAP GET request looks like:

```
REQ:
GET coaps://[2001:db8::2:1]:61616/est/att
(Content-Format: TBD6)
```

A 2.05 Content response contains attributes which are relevant for the authenticated client. In this example, the EST-coaps server returns two attributes that the client can ignore when they are unknown to him.

### A.3. enroll / reenroll

During the Enroll/Reenroll exchange, the EST-coaps client uses a CSR (Content-Format TBD7) request in the POST request payload.

After verification of the CSR by the server, a 2.05 Content response with the issued certificate will be returned to the client. As described in Section 4.4, if the server is not able to provide a response immediately, it sends an empty ACK with response code 5.03 (Service Unavailabel) and the Max-Age option. See Figure 3 for an example exchange.

[EDNOTE: When redoing this example, given that proof of possession (POP) is also used, make sure it is obvious that the ChallengePassword attribute in the CSR is valid HMAC output. HMAC-REAL.]

```
POST [2001:db8::2:1]:61616/est/sen
(token 0x45)
(Content-Format: TBD7)
h'30208530206d020100301f311d301b06035504031314646 56d6f7374657034203
133363831343133353230206230 0d06092a620673410101050003204f0030204a
022041005d9f4dffd3c5949f646a9584367778560950b355c35b8e34726dd3764
54231734795b4c09b9c6d75d408311307a81f7adef7f5d241f7d5be85620c5d44
38bbb4242cf215c167f2ccf36c364ea2618a62f0536576369d6304e6a96877224
7d86824f079faac7a6f694cfda5b84c42087dc062d462190c525813f210a036a7
37b4f30d8891f4b75559fb72752453146332d51c937557716ccec624f5125c3a4
447ad3115020048113fef54ad554ee88af09a2583aac9024075113db4990b1786
b871691e0f02030100018701f06092a62067341090731121310 2b72724369722f
372b455975353054343 00d06092a62067341010505000320 4100441b40177a3a6
5501487735a8ad5d3827a4eaa867013920e2afcda87aa81733c7c0353be47e1bf
a7cda5176e7ccc6be22ae03498588d5f2de3b143f2b1a6175ec544e8e7625af6b
836fd4416894c2e55ea99c6606f69075d6d53475d410729aa6d806afbb9986caf
7b844b5b3e4545f19071865ada007060cad6db26a592d4a7bda7d586b68110962
17071103407553155cddc75481e272b5ed553a8593fb7e25100a6f7605085dab4
fc7e0731f0e7fe305703791362d5157e92e6b5c2e3edbcadb40'

RET:
(Content-Format: TBD2)(token =0x45)
2.01 Created
h'3020f806092a62067341070283293020e50201013100300b06092a62067341070
1830b3020c730206fc20102020115300d06092a62067341010505003 01b311930
17060355040313106573744578616d706c654341204e774e301e170d313330353
0393233313535335a170d3134303530393233313535335a301f311d301b060355
0403131464656d6f737465703420313336383134313335323020623 00d06092a6
20673410101050003204f0030204a022041005d9f4dffd3c5949f646a95843677
78560950b355c35b8e34726dd376454231734795b4c09b9c6d75d408311307a81
f7adef7f5d241f7d5be85620c5d4438bbb4242cf215c167f2ccf36c364ea2618a
62f0536576369d6304e6a968772247d86824f079faac7a6f694cfda5b84c42087
dc062d462190c525813f210a036a737b4f30d8891f4b75559fb72752453146332
d51c937557716ccec624f5125c3a4447ad3115020048113fef54ad554ee88af09
a2583aac9024075113db4990b1786b871691e0f020301000134b050300e060355
1d0f0101f104030204c1d0603551d0e04160414e81d0788aa2710304c5ecd4d1e
065701f0603551d230418301653112966e304761732fbfe6a2c823c300d06092a
62067341010505000320 41002910d86f2ffeeb914c046816871de601567d291b4
3fabee0f0e8ff81cea27302a7133e20e9d04029866a8963c7d14e26fbe8a0ab1b
77fbb1214bbcdc906fbc381137ec1de685f79406c3e416b8d82f97174bc691637
5a4e1c4bf744c7572b4b2c6bade9fb35da786392ee0d95e3970542565f3886ad6
7746d1b12484bb02616e63302dc371dc6006e431fb7c457598dd204b367b0b3d3
258760a303f1102db26327f929b7c5a60173e1799491b69150248756026b80553
171e4733ad3d13c0103100'
```

### A.4. serverkeygen

During this valid /serverkeygen exchange, the EST-coaps client authenticates itself using the certificate provided by the connected CA.

The initial DTLS handshake is identical to the enrollment example. The CoAP GET request looks like:

[EDNOTE: same comment as HMAC-REAL above applies.]

[EDNOTE: Suggestion to have only one example with complete encrypted payload (the short one) and point out the different fields. Update this example according to the agreed upon solution from Section 4.5. ]

POST coaps://192.0.2.1:8085/est/skg
(token 0xa5)
(Content-Format: TBD7)(Max-Age=120)

h'3020813020690201003 05b313e303c06035504031335736572766572 4b657947656e20726571206279206 36c69656e7420696e2064656d6f207374657020313220313336383134313935353119301706035504051310504944 3a576964676574205 34e3a3130302062300d06092a620673410101050003204f0030204a02204100f4 dfa6c03f7f2766b23776c333d2c0f9d1a7a6ee36d01499bbe6f075d1e38a57e98 ecc197f51b75228454b7f19652332de5e52e4a974c6ae34e1df80b33f15f47d3b cbf76116bb0e4d3e04a9651218a476a13fc186c2a255e4065ff7c271cff104e47 31fad53c22b21a1e5138bf9ad0187314ac39445949a48805392390e78c7659621 6d3e61327a534f5ea7721d2b1343c7362b37da502717cfc2475653c7a3860c5f4 0612a5db6d33794d755264b6327e3a3263b149628585b85e57e42f6b3277591b0 2030100018701f06092a6206734109073112131064467341586d4a6e6a6f6b427 4447672300d06092a62067341010505000320410047 2d11007e5a2b2c2023d47a 6d71d046c307701d8ebc9e47272713378390b4ee321462a3dbe54579f5a514f6f 4050af497f428189b63655d03a194ef729f101743e5d03fbc6ae1e84486d1300a f9288724381909188c851fa9a5059802eb64449f2a3c9e441353d136768da27ff 4f277651d676a6a7e51931b08f56135a2230891fd184960e1313e7a1a9139ed19 28196867079a456cd2266cb754a45151b7b1b939e381be333fea61580fe5d25bf 4823dbd2d6a98445b46305c10637e202856611'

RET:
2.01 Content (Content-Format: TBD8)
(token=0xa5)

[TBD5,
h'30213e020100300d06092a62067341010105000421283021240201000220410 03c0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274 dd01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a1 1bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c

```
   0c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d5
   45e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704
   efec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e31083010
   001022041004e6b3f78b7791d6377f33117c17844531c81111fb8000282816264
   915565bc7c3f3f643b537a2c69140a31c22550fa97e5132c61b74166b68626704
   260620333050f510096b6570f5880e7e1c15dc0ca6ce2b5f187e2325da14ab705
   ad004717f3b2f779127b5c535e0cee6a343b502722f2397a26126e0af606b5aa7
   f96313511c0b7eb26354f91b82269de62757e3def807a6afdf83ddcbb0614bb7c
   542e6975d6456554e7bd9988fbd1930cd44d0e01ee9182ca54539418653150254
   1ad1a2a11e5021040bfce554b642c29131e7d65455e83c5406d76771912f758f5
   ee3ee36af386f38ffa313c0f661880c5a2b0970485d36f528e7f77a2e55b4ad76
   1242d1c2f75939c8061217d31491d305d3e07d6161c43e26f7de4477b1811de92
   33dc75b426302104015bf48ac376f52887813461fc54635517bcb67293837053e
   8ce1a33da7a35565a75a370dc14555b5316cb55742380350774d769d151ff0456
   0214389a232a2258326163167504cfce44cd316f63bb8a52da53a4cb74fd87194
   c0844881f791f23b0813ea0921325edd14459d41c8a1593f04316388e40b35fef
   7d2a195a5930fa54774427ac821eee2c62790d2c17bd192af794c611011506557
   83d4efe22185cbd83368786f2b1e68a5a27067e321066f0217b4b6d7971a3c21a
   241366b7907187583b511102103369047e5cce0b65012200df5ec697b5827575c
   db6821ff299d6a69574b31ddf0fbe9245ea2f74396c24b3a7565067e41366423b
   5bdd2b2a78194094dbe333f493d159b8e07722f2280d48388db7f1c9f0633bb0e
   173de2c3aa1f200af535411c7090210401421e2ea217e37312dcc606f453a6634
   f3df4dc31a9e910614406412e70eec9247f10672a500947a64356c015a845a7d1
   50e2e3911a2b3b61070a73247166da10bb45474cc97d1ec2bc392524307f35118
   f917438f607f18181684376e13a39e07',
   TBD2,
   h'3020c506092a62067341070283363020f20201013100300b06092a62067341070
   183183020d430207cc20102020116300d06092a62067341010505003 01b311930
   17060355040313106573744578616d706c654341204e774e301e170d313330353
   0393233323535365a170d3134303530393233323535365a302c312a3028060355
   040313217365727665727369646520 6b65792067656e6572617465642072657 37
   06f6e7365302062300d06092a62067341010105000 3204f0030204a022041003c
   0bc2748f2003e3e8ea15f746f2a71e83f585412b92cf6f8e64de02e056153274d
   d01c95dd9cff3112aa141774ab655c3d56359c3b3df055294692ed848e7e30a11
   bf14e47e0693d93017022b4cdb3e6d40325356152b213c8b535851e681a7074c0
   c6d2b60e7c32fc0336b28e743eba4e5921074d47195d3c05e43c527526e692d54
   5e562578d2d4b5f2191bff89d3eef0222764a2674637a1f99257216647df6704e
   fec5adbf54dab24231844eb595875795000e673dd6862310a146ad7e310830100
   0134b050300e0603551d0f0101f104030204c1d0603551d0e04160414764b1bd5
   e69935626e476b195a1a8c1f0603551d230418301653112966e304761732fbfe6
   a2c823c300d06092a62067341010505000 3204100474e5100a9cdaaa813b30f48
   40340fb17e7d6d6063064a5a7f2162301c464b5a8176623dfb1a4a484e618de1c
   3c3c5927cf590f4541233ff3c251e772a9a3f2c5fc6e5ef2fe155e5e385deb846
   b36eb4c3c7ef713f2d137ae8be4c022715fd033a818d55250f4e6077718180755
   a4fa677130da60818175ca4ab2af1d15563624c51e13dfdcf381881b72327e2f4
   9b7467e631a27b5b5c7d542bd2edaf78c0ac294f3972278996bdf673a334ff74c
   84aa7d65726310252f6a4f41281ec10ca2243864e3c5743103100']
```

Without the DecryptKeyIdentifier attribute, the response has no additional encryption beyond DTLS.

The response contains first a preamble that can be ignored. The EST-coaps server can use the preamble to include additional explanations, like ownership or support information

## Appendix B. EST-coaps Block message examples

Two examples are presented: (1) a cacerts exchange shows the use of Block2 and the block headers, and (2) a enroll exchange shows the Block1 and Block2 size negotiation for request and response payloads.

### B.1. cacerts block example

This section provides a detailed example of the messages using DTLS and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is the example value assumed for the DTLS datagram size. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a valid /cacerts exchange over DTLS. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 2509 bytes in binary. The CoAP message adds around 10 bytes, the DTLS record 29 bytes. To avoid IP fragmentation, the CoAP block option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 39 packets with a payload of 64 bytes each, followed by a packet of 13 bytes. The client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request 40 times. The server returns an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP response. The CoAP request-response exchange with block option is shown below. Block option is shown in a decomposed way (block-option:NUM/M/size) indicating the kind of Block option (2 in this case because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 in lock2 response means 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 with confirmable (CON) option and the content format of the Response is /application/cacerts.

```
   GET /192.0.2.1:8085/est/crts   (2:0/0/64)    -->
                <--   (2:0/1/64) 2.05 Content
   GET /192.0.2.1:8085/est/crts   (2:1/0/64)    -->
                <--   (2:1/1/64) 2.05 Content
                        |
                        |
                        |
   GET /192.0.2.1:8085/est/crts    (2:39/0/64)  -->
                <--   (2:39/0/64) 2.05 Content
```

40 blocks have been sent with partially filled block NUM=39 as last block.

For further detailing the CoAP headers, the first two blocks are written out.

The header of the first GET looks like:

```
  Ver = 1
  T = 0 (CON)
  Code = 0x01 (0.1 GET)
  Token = 0x9a    (client generated)
  Options
   Option1 (Uri-Host)            [optional]
     Option Delta = 0x3  (option nr = 3)
     Option Length = 0x9
     Option Value = 192.0.2.1
   Option2 (Uri-Port)            [optional]
     Option Delta = 0x4   (option nr = 3+4=7)
     Option Length = 0x4
     Option Value = 8085
   Option3 (Uri-Path)
     Option Delta = 0x4    (option nr = 7+4=11)
     Option Length = 0x5
     Option Value = "est"
   Option4 (Uri-Path)
     Option Delta = 0x0    (option nr = 11+0=11)
     Option Length = 0x6
     Option Value = "crts"
  Payload = [Empty]
```

The header of the first response looks like:

```
    Ver = 1
    T = 2 (ACK)
    Code = 0x45 (2.05 Content)
    Token = 0x9a     (copied by server)
    Options
      Option1 (Content-Format)
        Option Delta = 0xC  (option nr =12)
        Option Length = 0x2
        Option Value = TBD2
      Option2 (Block2)
        Option Delta = 0xB  (option 23 = 12 + 11)
        Option Length = 0x1
        Option Value = 0x0A (block number = 0, M=1, SZX=2)
    Payload =
  h'30233906092a6206734107028c2a3023260201013100300b06092a6206734107018
  c0c3020bb302063c20102020900a61e75193b7acc0d06092a6206734101'

  The second Block2:

    Ver = 1
    T = 2 (means ACK)
    Code = 0x45 (2.05 Content)
    Token = 0x9a     (copied by server)
    Options
      Option1 (Content-Format)
        Option Delta = 0xC  (option nr =12)
        Option Length = 0x2
        Option Value = TBD2
      Option2 (Block2)
        Option Delta = 0xB  (option 23 = 12 + 11)
        Option Length = 0x1
        Option Value = 0x1A (block number = 1, M=1, SZX=2)
    Payload =
  h'05050030
  1b31193017060355040313106573744578616d706c654341204f774f301e170d313
  330353039033353333315a170d3134303530393033353333315a'

  The 40th and final Block2:
```

```
Ver = 1
T = 2 (means ACK)
Code = 0x45      (2.05 Content)
Token = 0x9a     (copied by server)
Options
  Option1 (Content-Format)
    Option Delta = 0xC  (option nr =12)
    Option Length = 0x2
    Option Value = TBD2
  Option2 (Block2)
    Option Delta = 0xB  (option 23 = 12 + 11)
    Option Length = 0x2
    Option Value = 0x272 (block number = 39, M=0, SZX=2)
Payload = h'73a30d0c006343116f58403100'
```

## B.2. enroll block example

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 request/response consists of two parts: part1 containing the CSR transferred to the server, and part2 contains the certificate transferred back to the client. 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 Cert response takes N2+1 blocks. The header fields and the payload are omitted to show the block exchange. The type of payload is shown within curly brackets.

```
   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 enrolment with multiple blocks

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

## Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Michael C. Richardson
Sandelman Software Works

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

Martin Furuhed
Nexus Group

Email: martin.furuhed@nexusgroup.com

Shahid Raza
RISE SICS
Isafjordsgatan 22
Kista, Stockholm 16440
SE

Email: shahid@sics.se

ACE
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2018

M. Jones
Microsoft
L. Seitz
RISE SICS
G. Selander
Ericsson AB
S. Erdtman
Spotify
H. Tschofenig
ARM Ltd.
June 29, 2018

# Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)
# draft-ietf-ace-cwt-proof-of-possession-03

## 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.

## 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 31, 2018.

## Copyright Notice

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 [RFC7159] 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
                e68449c65f885d1b73b49eae1A0B0C0D0E0F10'
    }
```

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

The following example CWT Claims Set of a CWT (using 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'61626304050607080 90a0b0c0d0e0f10'
```

### 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.

### 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. Proof of possession can be used by recipients to reject messages from unauthorized senders. Audience restriction can be used by recipients to reject messages intended for different recipients.

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

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 [RFC5226] basis after a three-week review period on the cwt-reg-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Experts may approve registration once they are satisfied that such a specification will be published. [[ Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: cwt-reg-review@ietf.org. ]]

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

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

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

Expert, that Expert should defer to the judgment of the other Experts.

### 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): map
- Change Controller: IESG
- Specification Document(s): Section 3.2 of [[ this document ]]

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

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

## 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>.

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

[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>.

[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-12 (work in progress), May 2018.

[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/RFC7159, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

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

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

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

## 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 ]]

-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: September 6, 2018

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

March 05, 2018

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

## Abstract

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on September 6, 2018.

Table of Contents



## 1. Introduction

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

with Raw Public Keys (RPK) [RFC7250] and the DTLS handshake with Pre-Shared Keys (PSK) [RFC4279].

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

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

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

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

## 1.1. Terminology

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

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

# 2. Protocol Overview

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

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

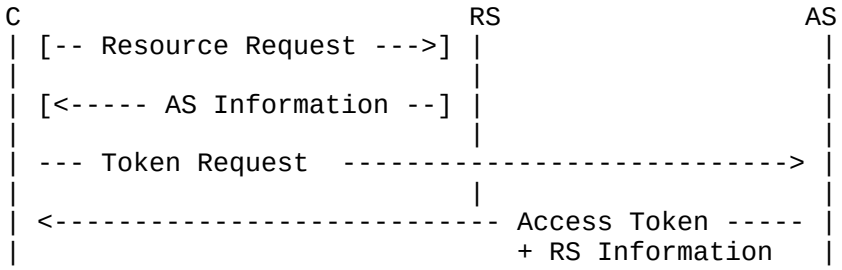



Figure 1: Retrieving an Access Token

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

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

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

The Access Token returned by the authorization server then can be used by the client to establish a new DTLS session with the resource server. When the client intends to use asymmetric cryptography in the DTLS handshake with the resource server, the client MUST upload

the Access Token to the authz-info resource on the resource server before starting the DTLS handshake, as described in section 5.8.1 of draft-ietf-ace-oauth-authz [4]. If only symmetric cryptography is used between the client and the resource server, the Access Token MAY instead be transferred in the DTLS ClientKeyExchange message (see Section 4.1).

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

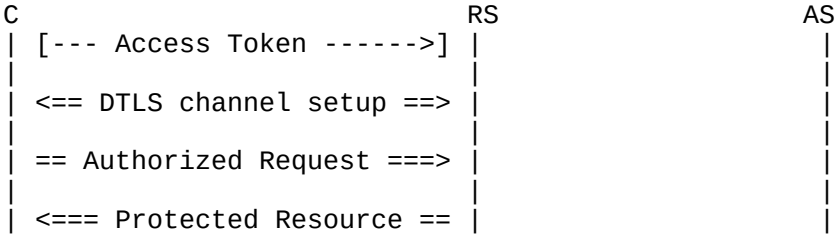



Figure 2: Protocol overview

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

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

## 2.1. Resource Access

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

On the resource server side, successful establishment of the DTLS channel binds the client to the access token, functioning as a proof-of-possession associated key. Any request that the resource server receives on this channel MUST be checked against these authorization rules that are associated with the identity of the client. Incoming

CoAP requests that are not authorized with respect to any Access Token that is associated with the client MUST be rejected by the resource server with 4.01 response as described in Section 5.1.1 of draft-ietf-ace-oauth-authz [5].

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

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

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

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

3. The request is destined for the resource server.

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

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

Incoming CoAP requests received on a secure DTLS channel MUST be rejected according to [Section 5.1.1 of draft-ietf-ace-oauth-authz](https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-10#section-5.1.1

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and

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

The client cannot always know a priori if an Authorized Resource Request will succeed. If the client repeatedly gets error responses containing AS Information (cf. Section 5.1.1 of draft-ietf-ace-oauth-authz [6] as response to its requests, it SHOULD request a new Access Token from the authorization server in order to continue communication with the resource server.

## 2.2. Dynamic Update of Authorization Information

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

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

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

Note: By associating the access tokens with the identifier of an existing DTLS session, the authorization information can be updated without changing the cryptographic keys for the DTLS communication between the client and the resource server, i.e. an existing session can be used with updated permissions.

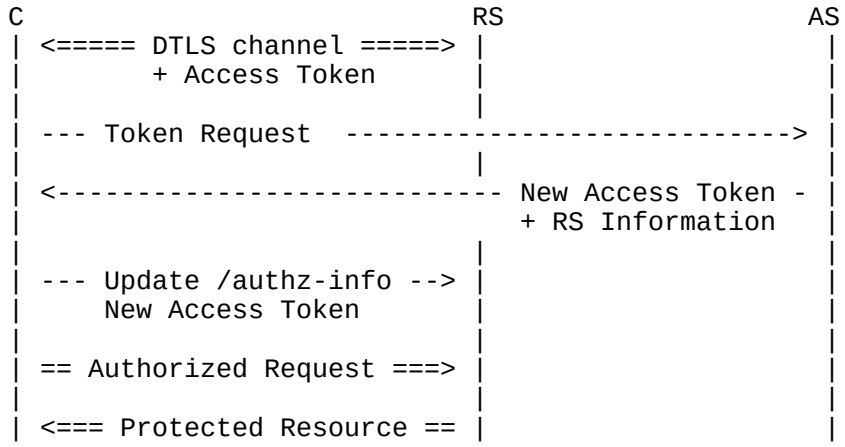



Figure 3: Overview of Dynamic Update Operation

## 2.3. Token Expiration

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

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

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

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

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

Table 1: Updated AS Information parameters

## 3. RawPublicKey Mode

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

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

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

Figure 4: Access Token Request Example for RPK Mode

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

When the authorization server authorizes a request, it will return an Access Token and a "cnf" object in the AS-to-Client response. Before

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

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

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

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

## 4. PreSharedKey Mode

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

Depending on the requested token type and algorithm in the Access Token request, the authorization server adds RS Information to the

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

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

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

Figure 5: Example Access Token response

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

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

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

Figure 6: Example Access Token response with reject

## 4.1. DTLS Channel Setup Between C and RS

When a client receives an Access Token from an authorization server, it checks if the payload contains an "access_token" parameter and a "cnf" parameter. With this information the client can initiate establishment of a new DTLS channel with a resource server. To use DTLS with pre-shared keys, the client follows the PSK key exchange algorithm specified in Section 2 of [RFC4279] using the key conveyed in the "cnf" parameter of the AS response as PSK when constructing the premaster secret.

In PreSharedKey mode, the knowledge of the session key by the client and the resource server is used for mutual authentication between both peers. Therefore, the resource server must be able to determine the session key from the Access Token. Following the general ACE authorization framework, the client can upload the Access Token to the resource server’s authz-info resource before starting the DTLS handshake. Alternatively, the client MAY provide the most recent Access Token in the "psk_identity" field of the ClientKeyExchange message. To do so, the client MUST treat the contents of the "access_token" field from the AS-to-Client response as opaque data and not perform any re-coding.

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

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

If no key with a matching identifier is found the resource server the resource server MAY process the decoded contents of the "psk_identity" field as access token that is stored with the authorization information endpoint before continuing the DTLS handshake. If the decoded contents of the "psk_identity" do not

yield a valid access token for the requesting client, the DTLS session setup is terminated with an "illegal_parameter" DTLS alert message.

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

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

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

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

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

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

### 4.2. Updating Authorization Information

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

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

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

## 5. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [I-D.ietf-ace-oauth-authz]. As it follows this framework's general approach, the general security and privacy considerations from section 6 and section 7 also apply to this profile.

Constrained devices that use DTLS [RFC6347] are inherently vulnerable to Denial of Service (DoS) attacks as the handshake protocol requires creation of internal state within the device. This is specifically of concern where an adversary is able to intercept the initial cookie exchange and interject forged messages with a valid cookie to continue with the handshake.

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

## 6. Privacy Considerations

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

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

## 7. IANA Considerations

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

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

Profile name: coap_dtls

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

Profile ID: 1

Change Controller: IESG

Reference: [RFC-XXXX]

## 8. References

### 8.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-10 (work in progress), February 2018.

[I-D.tiloca-tls-dos-handshake]
Tiloca, M., Seitz, L., Hoeve, M., and O. Bergmann, "Extension for protecting (D)TLS handshakes against Denial of Service", draft-tiloca-tls-dos-handshake-01 (work in progress), October 2017.

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

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

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

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

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

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

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

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

## 8.2. Informative References

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

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

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <https://www.rfc-editor.org/info/rfc6655>.

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

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

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

## 8.3. URIs

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

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

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

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

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

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

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

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

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

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

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

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

## Authors' Addresses

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

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

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

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

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

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

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

L. Seitz
RISE SICS
G. Selander
Ericsson
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
Arm Ltd.

July 2, 2018

# Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth) draft-ietf-ace-oauth-authz-13

## Abstract

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

## Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Table of Contents

## 1. Introduction

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

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

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

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

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

More detailed, interoperable specifications can be found in profiles. Implementations may claim conformance with a specific profile, whereby implementations utilizing the same profile interoperate while implementations of different profiles are not expected to be interoperable. Some devices, such as mobile phones and tablets, may implement multiple profiles and will therefore be able to interact with a wider range of low end devices. Requirements on profiles are described at contextually appropriate places throughout this specification, and also summarized in Appendix C.

## 2. Terminology

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

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

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

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors], such as client (C), resource server (RS), and authorization server (AS).

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as token and introspection at the AS and authz-info at the RS (see Section 5.8.1 for a definition of the authz-info endpoint). The CoAP [RFC7252] definition, which is "An entity participating in the CoAP protocol" is not used in this specification.

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

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

We use the term "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).

## 3. Overview

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

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

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

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

A fourth building block is the compact CBOR-based secure message format COSE [RFC8152], which enables application layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC5246]). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which is an extension to the OAuth tokens. The default token format is defined in CBOR web token (CWT) [RFC8392]. Application layer security for CoAP using COSE can be provided with OSCORE [I-D.ietf-core-object-security].

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

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

### 3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain scoped access to a resource with the permission of a resource owner. Authorization information, or references to it, is passed between the

nodes using access tokens. These access tokens are issued to clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

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

The token and introspection Endpoints:
: The AS hosts the token endpoint that allows a client to request access tokens. The client makes a POST request to the token endpoint on the AS and receives the access token in the response (if the request was successful).
In some deployments, a token introspection endpoint is provided by the AS, which can be used by the RS if it needs to request additional information regarding a received access token. The RS makes a POST request to the introspection endpoint on the AS and receives information about the access token in the response. (See "Introspection" below.)

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

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

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

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

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

Symmetric PoP key:
The AS generates a random symmetric PoP key. The key is either stored to be returned on introspection calls or encrypted and included in the access token. The PoP key is also encrypted for the client and sent together with the access token to the client.

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

The access token is either a simple reference, or a structured information object (e.g., CWT [RFC8392]) protected by a cryptographic wrapper (e.g., COSE [RFC8152]). The choice of PoP key does not necessarily imply a specific credential type for the integrity protection of the token.

Scopes and Permissions:
In OAuth 2.0, the client specifies the type of permissions it is seeking to obtain (via the scope parameter) in the access token request. In turn, the AS may use the scope response parameter to inform the client of the scope of the access token issued. As the client could be a constrained device as well, this specification defines the use of CBOR encoding as data format, see Section 5, to request scopes and to be informed what scopes the access token actually authorizes.

The values of the scope parameter in OAuth 2.0 are expressed as a list of space-delimited, case-sensitive strings, with a semantic that is well-known to the AS and the RS. More details about the concept of scopes is found under Section 3.3 in [RFC6749].

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

  While the structure and encoding of the access token varies throughout deployments, a standardized format has been defined with the JSON Web Token (JWT) [RFC7519] where claims are encoded as a JSON object. In [RFC8392], an equivalent format using CBOR encoding (CWT) has been defined.

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

## 3.2. CoAP

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

While HTTP uses headers and query strings to convey additional information about a request, CoAP encodes such information into header parameters called ’options’.

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

Transport layer security for CoAP can be provided by DTLS 1.2 [RFC6347] or TLS 1.2 [RFC5246]. CoAP defines a number of proxy operations that require transport layer security to be terminated at the proxy. One approach for protecting CoAP communication end-to-end through proxies, and also to support security for CoAP over a

different transport in a uniform way, is to provide security at the application layer using an object-based security mechanism such as COSE [RFC8152].

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

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

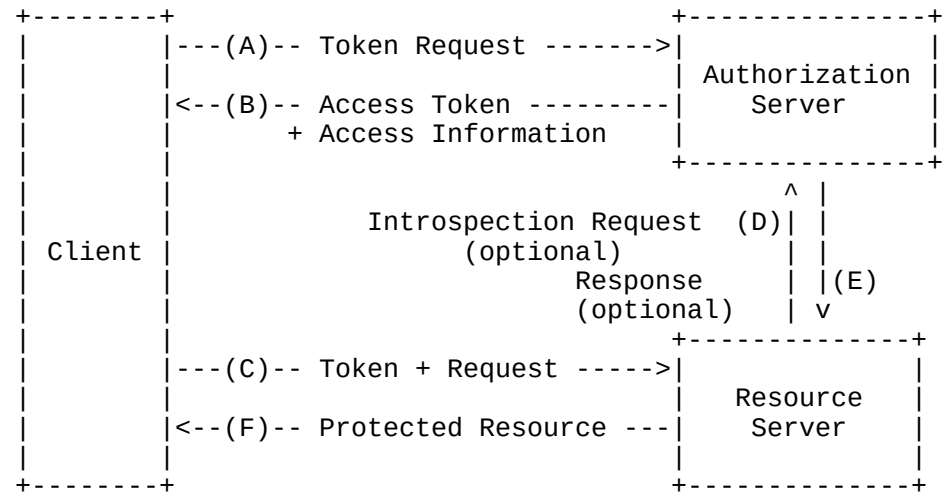



Figure 1: Basic Protocol Flow.

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. 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 and integrity protected. It is furthermore REQUIRED that the AS and the endpoint communicating with it (client or RS) perform mutual authentication.

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

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

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

### 5.1. Discovering Authorization Servers

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

Instead of the initial Unauthorized Resource Request message, 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 Information to enable the Client to request an access token from RS's AS as described in Section 5.1.2.

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

### 5.1.2. AS Information

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

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

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

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

Figure 2: AS Information parameters

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

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

Figure 3: AS Information payload example

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

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

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

Figure 4: AS Information example encoded in CBOR

## 5.2. Authorization Grants

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

The OAuth framework [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 the Content-Type and wrapping of the payload. The content of the request consists of the parameters specified in Section 4 of the OAuth 2.0 specification [RFC6749].

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

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

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

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

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

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

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

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

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

Figure 6 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example COSE is used to provide object-security, therefore the Content-Type is "application/cose".

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


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

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

Figure 7 shows a request for a token where a previously communicated proof-of-possession key is only referenced. Note that a transport layer based communication security profile with a CBOR payload is assumed in this example, therefore the Content-Type is "application/cbor". Also note that the client performs a password based authentication in this example by submitting its client_secret (see Section 2.3.1 of [RFC6749]).

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

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

### 5.6.2. AS-to-Client Response

If the access token request has been successfully verified by the AS and the client is authorized to obtain an access token corresponding to its access token request, the AS sends a response with the response code equivalent to the CoAP response code 2.01 (Created). If client request was invalid, or not authorized, the AS returns an error response as described in Section 5.6.3.

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

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

profile:
: OPTIONAL. This indicates the profile that the client MUST use towards the RS. See Section 5.6.4.4 for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

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

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

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

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

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

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

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

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key. Note that transport layer

security with CBOR encoding is assumed in this example, therefore the Content-Type is "application/cbor".

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

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

### 5.6.3. Error Response

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

- The Content-Type MUST be specified by the communication security profile used between client and AS. When using CoAP the raw payload before being processed by the communication security protocol MUST be encoded as a CBOR map.
- A response code equivalent to the CoAP code 4.00 (Bad Request) MUST be used for all error responses, except for invalid_client where a response code equivalent to the CoAP code 4.01 (Unauthorized) MAY be used under the same conditions as specified in Section 5.2 of [RFC6749].
- The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12, when a CBOR encoding is used.
- The error code (i.e., value of the "error" parameter) MUST be abbreviated as specified in Figure 10, when a CBOR encoding is used.

| Name | CBOR Values |
|---|---|
| invalid_request | 0 |
| invalid_client | 1 |
| invalid_grant | 2 |
| unauthorized_client | 3 |
| unsupported_grant_type | 4 |
| invalid_scope | 5 |
| unsupported_pop_key | 6 |

Figure 10: CBOR abbreviations for common error codes

In addition to the error responses defined in OAuth 2.0, the following behavior MUST be implemented by the AS: If the client submits an asymmetric key in the token request that the RS cannot process, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "unsupported_pop_key" defined in Figure 10.

#### 5.6.4. Request and Response Parameters

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

##### 5.6.4.1. Audience

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

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

##### 5.6.4.2. Grant Type

The abbreviations in Figure 11 MUST be used in CBOR encodings instead of the string values defined in [RFC6749], if CBOR payloads are used.

| Name | CBOR Value | Original Specification |
|---|---|---|
| password | 0 | RFC6749 |
| authorization_code | 1 | RFC6749 |
| client_credentials | 2 | RFC6749 |
| refresh_token | 3 | RFC6749 |

Figure 11: CBOR abbreviations for common grant types

#### 5.6.4.3. Token Type

The "token_type" parameter, 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.

#### 5.6.4.4. Profile

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

A profile MUST specify an identifier that MUST be used to uniquely identify itself in the "profile" parameter. The textual representation of the profile identifier is just intended for human readability and MUST NOT be used in parameters and claims..

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

#### 5.6.4.5. Confirmation

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

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

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

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

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

### 5.6.5. Mapping Parameters to CBOR

If CBOR encoding is used, all OAuth parameters in access token requests and responses MUST be mapped to CBOR types as specified in Figure 12, using the given integer abbreviation for the map keys.

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

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

Figure 12: CBOR mappings used in token requests

## 5.7. The 'Introspect' Endpoint

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

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

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

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

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

#### 5.7.1. RS-to-AS Request

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

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

For example, Figure 13 shows a RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that object security based on COSE is assumed in this example, therefore the Content-Type is "application/cose+cbor". Figure 14 shows the decoded payload.

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "introspect"
Content-Type: "application/cose+cbor"
Payload:
... COSE content ...
```

Figure 13: Example introspection request.

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

Figure 14: Decoded token.

### 5.7.2. AS-to-RS Response

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

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

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

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

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

For example, Figure 15 shows an AS response to the introspection request in Figure 13. Note that transport layer security is assumed in this example, therefore the Content-Type is "application/cbor".

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

Figure 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, the Content-Type MUST be set according to the specification of the communication security profile. If CoAP is used the payload MUST be encoded as a CBOR map.
- If the credentials used by the RS are invalid the AS MUST respond with the response code equivalent to the CoAP code 4.01 (Unauthorized) and use the required and optional parameters from Section 5.2 in RFC 6749 [RFC6749].
- If the RS does not have the right to perform this introspection request, the AS MUST respond with a response code equivalent to the CoAP code 4.03 (Forbidden). In this case no payload is returned.
- The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12.
- The error codes MUST be abbreviated using the codes specified in Figure 10.

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

#### 5.7.4. Mapping Introspection parameters to CBOR

If CBOR is used, the introspection request and response parameters MUST be mapped to CBOR types as specified in Figure 16, using the given integer abbreviation for the map key.

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

| Parameter name | CBOR Key | Value Type |
|---|---|---|
| iss | 1 | text string |
| sub | 2 | text string |
| aud | 3 | text string |
| exp | 4 | integer or floating-point number |
| nbf | 5 | integer or floating-point number |
| iat | 6 | integer or floating-point number |
| cti | 7 | byte string |
| client_id | 8 | text string |
| scope | 12 | text OR byte string |
| token_type | 20 | text string |
| username | 22 | text string |
| cnf | 25 | map |
| profile | 26 | unsigned integer |
| token | 27 | byte string |
| token_type_hint | 28 | text string |
| active | 29 | True or False |
| rs_cnf | 30 | map |

Figure 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 draft uses the "cnf" claim from [I-D.ietf-ace-cwt-proof-of-possession] and specifies the "scope" claim for both JSON and CBOR web tokens.

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

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

If the AS needs to convey a hint to the RS about which profile it should use to communicate with the client, the AS MAY include a

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

#### 5.8.1. The 'Authorization Information' Endpoint

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

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

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

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

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

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

Profiles MUST specify how the authz-info endpoint is protected, including how error responses from this endpoint are protected. Note that since the token contains information that allow the client and

the RS to establish a security context in the first place, mutual authentication may not be possible at this point.

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

#### 5.8.2. Client Requests to the RS

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

The response code MUST be 4.01 (Unauthorized) in case the client has not performed the proof-of-possession, or if RS has no valid access token for the client. If RS has an access token for the client but not for the resource that was requested, RS MUST reject the request with a 4.03 (Forbidden). If RS has an access token for the client but it does not cover the action that was requested on the resource, RS MUST reject the request with a 4.05 (Method Not Allowed).

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

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

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

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

#### 5.8.3. Token Expiration

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

- The token is a CWT and includes an "exp" claim and possibly the "nbf" claim. The RS verifies these by comparing them to values

from its internal clock as defined in [RFC7519]. In this case the RS's internal clock must reflect the current date and time, or at least be synchronized with the AS's clock. How this clock synchronization would be performed is out of scope for this specification.

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

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

## 6. Security Considerations

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

A large range of threats can be mitigated by protecting the contents of the access token by using a digital signature or a keyed message digest (MAC) or an Authenticated Encryption with Associated Data (AEAD) algorithm. Consequently, the token integrity protection MUST be applied to prevent the token from being modified, particularly since it contains a reference to the symmetric key or the asymmetric key. If the access token contains the symmetric key, this symmetric key MUST be encrypted by the authorization server so that only the resource server can decrypt it. Note that using an AEAD algorithm is preferable over using a MAC unless the message needs to be publicly readable.

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

The authorization server MUST offer confidentiality protection for any interactions with the client. This step is extremely important since the client may obtain the proof-of-possession key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby completely negating proof-of-possession security. Profiles MUST specify how confidentiality protection is provided, and additional protection can be applied by encrypting the token, for example encryption of CWTs is specified in Section 5.1 of [RFC8392].

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

Clients can at any time request a new proof-of-possession capable access token. If clients have that capability, the AS can keep the lifetime of the access token and the associated proof-of-possession key short and therefore use shorter proof-of-possession key sizes, which translate to a performance benefit for the client and for the resource server. Shorter keys also lead to shorter messages (particularly with asymmetric keying material).

When authorization servers bind symmetric keys to access tokens, they SHOULD scope these access tokens to a specific permissions. Furthermore access tokens using symmetric keys for proof-of-possession SHOULD NOT be targeted at an audience that contains more than one RS, since otherwise any RS in the audience that receives that access token can impersonate the client towards the other members of the audience.

### 6.1. Unprotected AS Information

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the AS information contained in an unprotected response from RS to an unauthorized request (see Section 5.1.2) is authentic. It is therefore advisable to provide C with a (possibly hard-coded) list of trustworthy

authorization servers. AS information responses referring to a URI not listed there would be ignored.

### 6.2. Use of Nonces for Replay Protection

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

### 6.3. Combining profiles

There may 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.

### 6.4. Error responses

The various error responses defined in this framework may leak information to an adversary. For example errors responses for requests to the Authorization Information endpoint can reveal information about an otherwise opaque access token to an adversary who has intercepted this token. This framework is written under the assumption that, in general, the benefits of detailed error messages outweigh the risk due to information leakage. For particular use cases, where this assessment does not apply, detailed error messages can be replaced by more generic ones.

## 7. Privacy Considerations

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

The AS is in a very central position and can potentially learn sensitive information about the clients requesting access tokens. If the client credentials grant is used, the AS can track what kind of access the client intends to perform. With other grants this can be prevented by the Resource Owner. To do so, the resource owner needs to bind the grants it issues to anonymous, ephemeral credentials that do not allow the AS to link different grants and thus different access token requests by the same client.

If access tokens are only integrity protected and not encrypted, they may reveal information to attackers listening on the wire, or able to acquire the access tokens in some other way. In the case of CWTs the token may, e.g., reveal the audience, the scope and the confirmation method used by the client. The latter may reveal the identity of the device or application running the client. This may be linkable to the identity of the person using the client (if there is a person and not a machine-to-machine interaction).

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

An unprotected response to an unauthorized request (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. Authorization Server Information

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

The columns of the registry are:

Name The name of the parameter

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

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

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

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

### 8.2. OAuth Error Code CBOR Mappings Registry

This section establish the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. It should be noted that, in addition to the expert review, some portions of the registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of the registry are:

Name The OAuth Error Code name, refers to the name in Section 5.2. of [RFC6749], e.g., "invalid_request".
CBOR Value CBOR abbreviation for this error code. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.

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

### 8.3. OAuth Grant Type CBOR Mappings

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

The columns of this registry are:

Name The name of the grant type as specified in Section 1.3 of [RFC6749].
CBOR Value CBOR abbreviation for this grant type. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action.

Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

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

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

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

### 8.4. OAuth Access Token Types

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

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

### 8.5. OAuth Token Type CBOR Mappings

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

The columns of this registry are:

Name The name of token type as registered in the OAuth Access Token Types registry, e.g., "Bearer".

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

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

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

#### 8.5.1. Initial Registry Contents

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

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

### 8.6. ACE Profile Registry

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

The columns of this registry are:

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

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

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

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

### 8.7. OAuth Parameter Registration

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

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

- Name: "profile"

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

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

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

### 8.8. OAuth CBOR Parameter Mappings Registry

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

The columns of this registry are:

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

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

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

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

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

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

### 8.9. OAuth Introspection Response Parameter Registration

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

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

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

### 8.10. Introspection Endpoint CBOR Mappings Registry

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

The columns of this registry are:

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

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

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

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

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

### 8.11. JSON Web Token Claims

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

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

### 8.12. CBOR Web Token Claims

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

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

## 9. Acknowledgments

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

Thanks to Eve Maler for her contributions to the use of OAuth 2.0 and UMA in IoT scenarios, Robert Taylor for his discussion input, and Malisa Vucinic for his input on the predecessors of this proposal.

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

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

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

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

# 10. References

## 10.1. Normative References

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

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

[IANA.JsonWebTokenClaims]
IANA, "JSON Web Token Claims", <https://www.iana.org/assignments/jwt/jwt.xhtml#claims>.

[IANA.OAuthAccessTokenTypes]
IANA, "OAuth Access Token Types", <https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-types>.

[IANA.OAuthParameters]
IANA, "OAuth Parameters", <https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#parameters>.

[IANA.TokenIntrospectionResponse]
IANA, "OAuth Token Introspection Response", <https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#token-introspection-response>.

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

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

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

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

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

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

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

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

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

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

## 10.2. Informative References

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

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-06 (work in progress), November 2017.

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-12 (work in progress), March 2018.

[I-D.ietf-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-10 (work in progress), June 2018.

[Margi10impact]
Margi, C., de Oliveira, B., de Sousa, G., Simplicio Jr, M., Barreto, P., Carvalho, T., Naeslund, M., and R. Gold, "Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds", Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), 2010 August.

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

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

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

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

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

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

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

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

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

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

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

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

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

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

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

[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>.

## 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 REQUIRES the use of CBOR [RFC7049] as data format. Where CBOR data needs to be protected, the use of COSE [RFC8152] is RECOMMENDED. Furthermore where self-contained tokens are needed, this framework RECOMMENDS the use of CWT [RFC8392]. These measures aim at reducing the size of messages sent over the wire, the RAM size of data objects that need to be kept in memory and the size of libraries that devices need to support.

CoAP:

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

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 includes the "profile" and the "rs_cnf" parameters. This aims at enabling scenarios, where a powerful client, supporting multiple profiles, needs to interact with a RS for which it does not know the supported profiles and the raw public key.

Proof-of-Possession:

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

Auth-Info endpoint:

This framework introduces a new way of providing access tokens to a RS by exposing a authz-info endpoint, to which access tokens can be POSTed. This aims at reducing the size of the request message and the code complexity at the RS. The size of the request message is problematic, since many constrained protocols have

severe message size limitations at the physical layer (e.g., in the order of 100 bytes). This means that larger packets get fragmented, which in turn combines badly with the high rate of packet loss, and the need to retransmit the whole message if one packet gets lost. Thus separating sending of the request and sending of the access tokens helps to reduce fragmentation.

Client Credentials Grant:

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

Introspection:

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

## Appendix B. Roles and Responsibilities

Resource Owner

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

Requesting Party

* Make sure that the client is provisioned the necessary credentials to authenticate to the AS.
* Make sure that the client is configured to follow the security requirements of the Requesting Party when issuing requests (e.g., minimum communication security requirements, trust anchors).
* Register the client at the AS. This includes making known to the AS which profiles, token_types, and key types (symmetric/ asymmetric) the client.

Authorization Server

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

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).

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

Resource Server

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

## Appendix C. Requirements on Profiles

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

- Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.6.4.4
- Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS over CoAP).

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

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

## Appendix E. Deployment Examples

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

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

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

### E.1. Local Token Validation

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

Since the resource server must be able to verify the access token locally, self-contained access tokens must be used.

This example shows the interactions between a client, the authorization server and a temperature sensor acting as a resource server. Message exchanges A and B are shown in Figure 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.

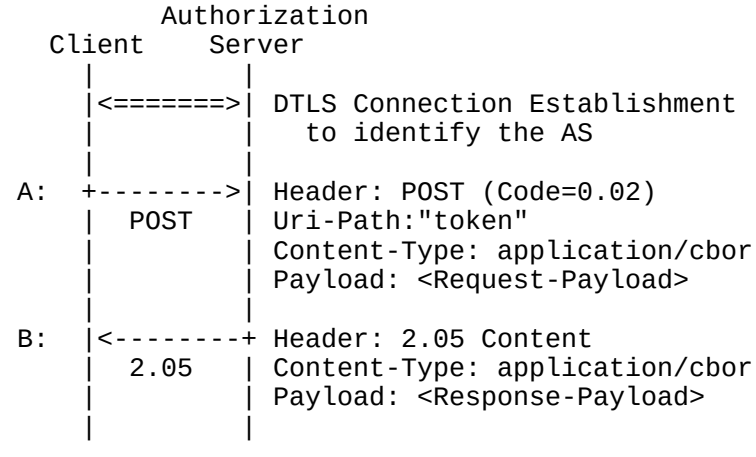



Figure 17: Token Request and Response Using Client Credentials.

The information contained in the Request-Payload and the Response-Payload is shown in Figure 18. Note that a TLS/DTLS-based communication security profile is used in this example. Hence, the Content-Type is "application/cbor".

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

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

Figure 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.

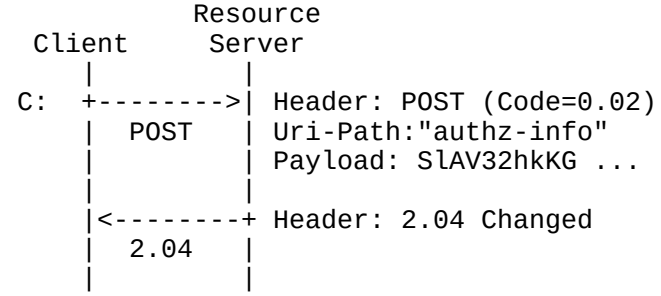



Figure 20: Access Token provisioning to RS

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.

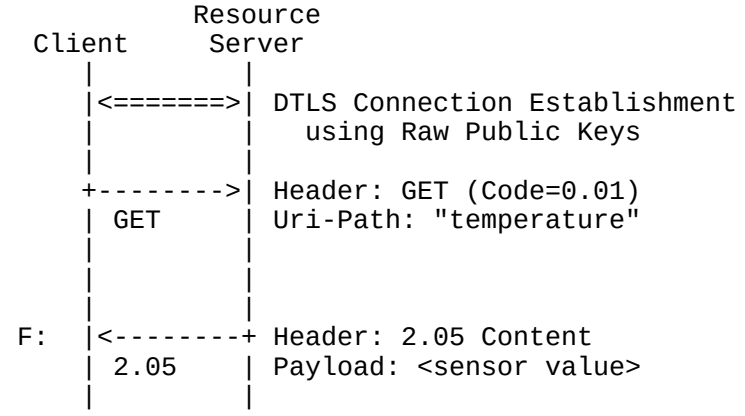



Figure 21: Resource Request and Response protected by DTLS.

### 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.

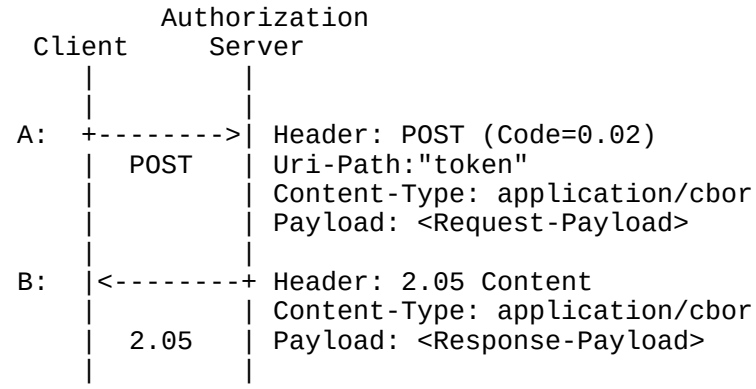



Figure 22: Token Request and Response using Client Credentials.

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

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

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

Figure 23: Request and Response Payload for C offline

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

C: Next, the client POSTs the access token to the authz-info endpoint in the RS. This is a plain CoAP request, i.e., no DTLS between client and RS. Since the token is an opaque string, the RS cannot verify it on its own, and thus defers to respond the client with a status code until after step E.
D: The RS forwards the token to the introspection endpoint on the AS. Introspection assumes a secure connection between the AS and the RS, e.g., using transport of application layer security. In the example AS is identified using pre shared security context (psk, rpk or certificate) while RS is acting as client and is identified with client_id and client_secret.
E: The AS provides the introspection response containing parameters about the token. This includes the confirmation key (cnf) parameter that allows the RS to verify the client's proof of possession in step F.
After receiving message E, the RS responds to the client's POST in step C with the CoAP response code 2.01 (Created).

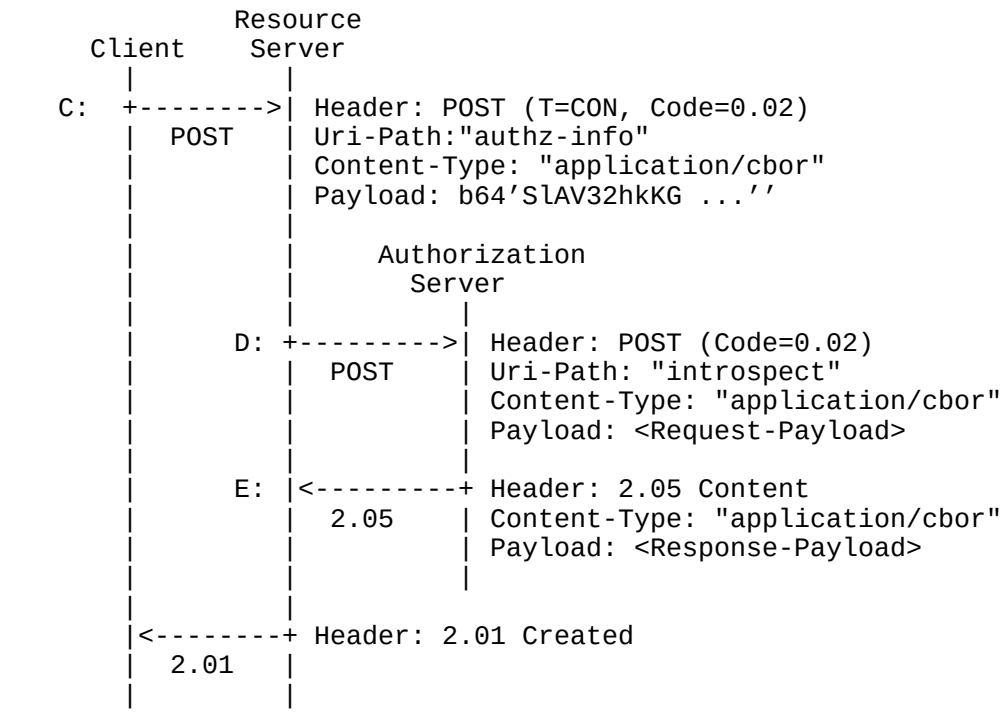



Figure 24: Token Introspection for C offline

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

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

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

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.

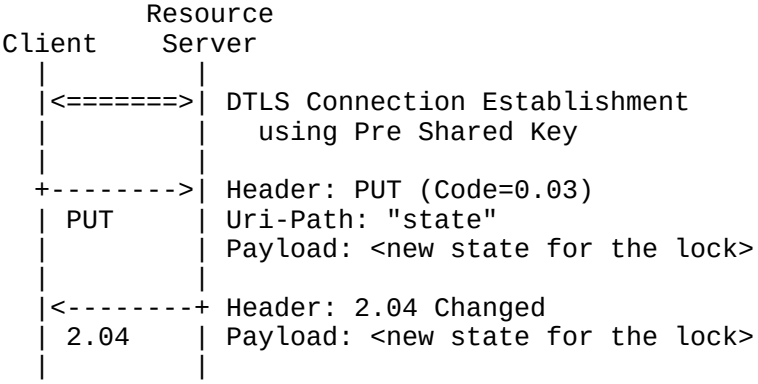



Figure 26: Resource request and response protected by OSCORE

## Appendix F. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

### F.1. Version -12 to -13

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

### F.2. 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.3. Version -10 to -11

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

### F.4. 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.5. Version -08 to -09

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

### F.6. 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.7. Version -06 to -07

- Various clarifications added.
- Fixed erroneous author email.

### F.8. 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.9. 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.10. 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.11. 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.12. 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.13. Version -00 to -01

- Changed 5.1. from "Communication Security Protocol" to "Client Information".
- Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP access token request profile for IoT.
  - Allow the client to indicate preferences for the communication security protocol.
  - Defined the term "Client Information" for the additional information returned to the client in addition to the access token.
  - Require that the messages between AS and client are secured, either with (D)TLS or with COSE_Encrypted wrappers.
  - Removed dependency on OSCOAP and added generic text about object security instead.
  - Defined the "rpk" parameter in the client information to transmit the raw public key of the RS from AS to client.

    * (D)TLS MUST use the PoP key in the handshake (either as PSK or as client RPK with client authentication).
    * Defined the use of x5c, x5t and x5tS256 parameters when a client certificate is used for proof of possession.
    * Defined "tktn" parameter for signaling for how to transfer the access token.
  - Added 5.2. the CoAP Access-Token option for transferring access tokens in messages that do not have payload.
  - 5.3.2. Defined success and error responses from the RS when receiving an access token.
  - 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
  - Appendix B Added list of roles and responsibilities for C, AS and RS.

Authors' Addresses

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

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

Email: erdtman@spotify.com

Hannes Tschofenig
Arm Ltd.
Absam 6067
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2018

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

June 28, 2018

# OSCORE profile of the Authentication and Authorization for Constrained Environments Framework
## draft-ietf-ace-oscore-profile-02

Abstract

This memo specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework. It utilizes Object Security for Constrained RESTful Environments (OSCORE) to provide communication security, server authentication, and proof-of-possession for a key owned by the client and bound to an OAuth 2.0 access token.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 30, 2018.

Table of Contents



## 1. Introduction

This memo specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use CoAP [RFC7252] to communicate. The client uses an access token, bound to a key (the proof-of-possession key) to authorize its access to the resource server. In order to provide communication security, proof of possession, and server authentication they use Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security]. Optionally the client and the resource server may also use CoAP and OSCORE to communicate with the authorization server.

OSCORE specifies how to use CBOR Object Signing and Encryption (COSE) [RFC8152] to secure CoAP messages. In order to provide replay and reordering protection OSCORE also introduces sequence numbers that are used together with COSE.

Note that OSCORE can be used to secure CoAP messages, as well as HTTP and combinations of HTTP and CoAP; a profile of ACE similar to the one described in this document, with the difference of using HTTP instead of CoAP as communication protocol, could be specified analogously to this one.

### 1.1. Terminology

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

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

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

Terminology for entities in the architecture is defined in OAuth 2.0 [RFC6749] and [I-D.ietf-ace-actors], such as client (C), resource server (RS), and authorization server (AS). It is assumed in this document that a given resource on a specific RS is associated to a unique AS.

## 2. Client to Resource Server

The use of OSCORE for arbitrary CoAP messages is specified in [I-D.ietf-core-object-security]. This section defines the specific uses and their purpose for securing the communication between a client and a resource server, and the parameters needed to negotiate the use of this profile with the token resource at the authorization server as specified in section 5.6 of [I-D.ietf-ace-oauth-authz].

### 2.1. Signaling the use of OSCORE

A client requests a token at an AS via the /token resource. This follows the message formats specified in section 5.6.1 of [I-D.ietf-ace-oauth-authz].

The AS responding to a successful access token request as defined in section 5.6.2 of [I-D.ietf-ace-oauth-authz] can signal that the use of OSCORE is REQUIRED for a specific access token by including the "profile" parameter with the value "coap_oscore" in the access token response. This means that the client MUST use OSCORE towards all

resource servers for which this access token is valid, and follow Section 2.2 to derive the security context to run OSCORE.

The error response procedures defined in section 5.6.3 of the ACE framework are unchanged by this profile.

Note the the client and the authorization server MAY OPTIONALLY use OSCORE to protect the interaction via the /token resource. See Section 3 for details.

## 2.2. Key establishment for OSCORE

Section 3.2 of [I-D.ietf-core-object-security] defines how to derive a security context based on a shared master secret and a set of other parameters, established between client and server. The proof-of-possession key (pop-key) provisioned from the AS MAY, in case of pre-shared keys, be used directly as master secret in OSCORE.

If OSCORE is used directly with the symmetric pop-key as master secret, then the AS MUST provision the following data, in response to the access token request:

- a master secret
- the sender identifier
- the recipient identifier

Additionally, the AS MAY provision the following data, in the same response. In case these parameters are omitted, the default values are used as described in section 3.2 of [I-D.ietf-core-object-security].

- an AEAD algorithm
- a KDF algorithm
- a salt
- a replay window type and size

The master secret MUST be communicated as COSE_Key in the 'cnf' parameter of the access token response as defined in Section 5.6.4.5 of [I-D.ietf-ace-oauth-authz]. The AEAD algorithm MAY be included as the 'alg' parameter in the COSE_Key; the KDF algorithm MAY be included as the 'kdf' parameter of the COSE_Key and the salt MAY be included as the 'slt' parameter of the COSE_Key as defined in table 1.

The same parameters MUST be included as metadata of the access token; if the token is a CWT [RFC8392], the same COSE_Key structure MUST be placed in the 'cnf' claim of this token. If a CWT is used it MUST be encrypted, since the token is transferred from the client to the RS over an unprotected channel.

The AS MUST also assign identifiers to both client and RS, which are then used as Sender ID and Recipient ID in the OSCORE context as described in section 3.1 of [I-D.ietf-core-object-security]. These identifiers MUST be unique in the set of all clients and RS identifiers for a certain AS. Moreover, these MUST be included in the COSE_Key as header parameters, as defined in table 1.

We assume in this document that a resource is associated to one single AS, which makes it possible to assume unique identifiers for each client requesting a particular resource to a RS. If this is not the case, collisions of identifiers may appear in the RS, in which case the RS needs to have a mechanism in place to disambiguate identifiers or mitigate their effect.

Note that C should set the Sender ID of its security context to the clientId value received and the Recipient ID to the serverId value, and RS should do the opposite.

| name | label | CBOR type | registry | description |
|---|---|---|---|---|
| clientId | TBD1 | bstr | | Identifies the client in an OSCORE context using this key |
| serverId | TBD2 | bstr | | Identifies the server in an OSCORE context using this key |
| kdf | TBD3 | bstr | | Identifies the KDF algorithm in an OSCORE context using this key |
| slt | TBD4 | bstr | | Identifies the master salt in an OSCORE context using this key |

Table 1: Additional COSE_Key Common Parameters

Figure 1 shows an example of such an AS response, in CBOR diagnostic notation without the tag and value abbreviations.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "alg" : "AES-CCM-16-64-128",
      "clientId" : b64'qA',
      "serverId" : b64'Qg',
      "k" : b64'+a+Dg2jjU+eIiOFCa9lObw'
    }
  }
}
```

Figure 1: Example AS response with OSCORE parameters.

Figure 2 shows an example CWT, containing the necessary OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "alg" : "AES-CCM-16-64-128",
      "clientId" : b64'Qg',
      "serverId" : b64'qA',
      "k" : b64'+a+Dg2jjU+eIiOFCa9lObw'
  }
}
```

Figure 2: Example CWT with OSCORE parameters.

Note that the proof-of-possession required to bind the access token to the client is implicitly performed by generating the shared OSCORE context using the pop-key as master secret, both on the client and RS side. 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.

## 3. Client to Authorization Server

As specified in the ACE framework (section 5.6 of [I-D.ietf-ace-oauth-authz]), the Client and AS can also use CoAP instead of HTTP to communicate via the token resource. This section specifies how to use OSCORE between Client and AS together with CoAP. The use of OSCORE for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The client and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the client and the AS communicate using OSCORE ([I-D.ietf-core-object-security]) through the introspection resource as specified in section 5.7 of [I-D.ietf-ace-oauth-authz].

## 4. Resource Server to Authorization Server

As specified in the ACE framework (section 5.7 of [I-D.ietf-ace-oauth-authz]), the RS and AS can also use CoAP instead of HTTP to communicate via the introspection resource. This section specifies how to use OSCORE between RS and AS. The use of OSCORE for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The RS and the AS are expected to have pre-established security contexts in place. How these security contexts are established is out of scope for this profile. Furthermore the RS and the AS communicate using OSCORE ([I-D.ietf-core-object-security]) through the introspection resource as specified in section 5.7 of [I-D.ietf-ace-oauth-authz].

## 5. 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.

## 6. Privacy Considerations

TBD.

## 7. IANA Considerations

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

The following registration is done for the ACE OAuth Profile Registry following the procedure specified in section 8.6 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]]

The following registrations are done for the COSE Key Common Parameter Registry specified in section 16.5 of [RFC8152]:

- Name: clientId
- Label: TBD1 (value between 1 and 255)
- CBOR Type: bstr
- Value Registry: N/A
- Description: Identifies the client in an OSCORE context
- Reference: [[this specification]]

- Name: serverId
- Label: TBD2 (value between 1 and 255)
- Value Type: bstr
- Value Registry: N/A
- Description: Identifies the server in an OSCORE context
- Reference: [[this specification]]

- Name: kdf
- Label: TBD3 (value between 1 and 255)
- Value Type: bstr
- Value Registry: COSE Algorithms registry
- Description: Identifies the KDF algorithm to be used in an OSCORE context

- Reference: [[this specification]]

- Name: slt
- Label: TBD4 (value between 1 and 255)
- Value Type: bstr
- Value Registry: N/A
- Description: Identifies the master salt of to be used in an OSCORE context
- Reference: [[this specification]]

# 8. References

## 8.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-12 (work in progress), May 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.

[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>.

[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.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-04 (work in progress), October 2015.

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-06 (work in progress), November 2017.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-08 (work in progress), March 2018.

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

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

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

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

## Appendix A. Profile Requirements

This section lists the specifications on this profile based on the requirements on the framework, as requested in Appendix C of [I-D.ietf-ace-oauth-authz].

- (Optional) discovery process of how the client finds the right AS for an RS it wants to send a request to: Not specified
- communication protocol the client and the RS must use: CoAP
- security protocol the client and RS must use: OSCORE
- how the client and the RS mutually authenticate: Implicitly by possession of a common OSCORE security context

- Content-format of the protocol messages: "application/cose+cbor"
- proof-of-possession protocol(s) and how to select one; which key types (e.g. symmetric/asymmetric) supported: OSCORE algorithms; pre-established symmetric keys
- profile identifier: coap_oscore
- (Optional) how the RS talks to the AS for introspection: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- how the client talks to the AS for requesting a token: HTTP/CoAP (+ TLS/DTLS/OSCORE)
- how/if the /authz-info endpoint is protected: Security protocol above
- (Optional)other methods of token transport than the /authz-info endpoint: no

## Appendix B. Using the pop-key with EDHOC (EDHOC+OSCORE)

EDHOC specifies an authenticated Diffie-Hellman protocol that allows two parties to use CBOR [RFC7049] and COSE in order to establish a shared secret key with perfect forward secrecy. The use of Ephemeral Diffie-Hellman Over COSE (EDHOC) [I-D.selander-ace-cose-ecdhe] in this profile in addition to OSCORE, provides perfect forward secrecy (PFS) and the initial proof-of-possession, which ties the proof-of-possession key to an OSCORE security context.

If EDHOC is used together with OSCORE, and the pop-key (symmetric or asymmetric) is used to authenticate the messages in EDHOC, then the AS MUST provision the following data, in response to the access token request:

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

How these parameters are communicated depends on the type of key (asymmetric or symmetric). Moreover, the AS MUST signal the use of OSCORE + EDHOC with the ’profile’ parameter set to "coap_oscore_edhoc" and follow Appendix B to derive the security context to run OSCORE.

Note that in the case described in this section, the ’expires_in’ parameter, defined in Section 4.2.2. of [RFC6749] defines the lifetime in seconds of both the access token and the shared secret. After expiration, C MUST acquire a new access token from the AS, and run EDHOC again, as specified in this section

### B.1. Using Asymmetric Keys

In case of an asymmetric key, C MUST communicate its own asymmetric key to the AS in the 'cnf' parameter of the access token request, as specified in Section 5.6.1 of [I-D.ietf-ace-oauth-authz]; the AS MUST communicate the RS's public key to C in the response, in the 'rs_cnf' parameter, as specified in Section 5.6.1 of [I-D.ietf-ace-oauth-authz]. Note that the RS's public key MUST include a 'kid' parameter, and that the value of the 'kid' MUST be included in the access token, to let the RS know which of its public keys C used. If the access token is a CWT [RFC8392], the key identifier MUST be placed directly in the 'cnf' structure (if the key is only referenced).

Figure 3 shows an example of such a request in CBOR diagnostic notation without tag and value abbreviations.

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

Figure 3: Example access token request (OSCORE+EDHOC, asymmetric).

Figure 4 shows an example of a corresponding response in CBOR diagnostic notation without tag and value abbreviations.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (contains "kid" : "client_key")',
  "profile" : "coap_oscore_edhoc",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kid" : "server_key"
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'cGJ90UiglWiGahtagnv8usWxHK2PmfnHKwXPS54m0kT',
      "y" : b64'reASjpkttcsz+1rb7btKLv8EX4IBOL+C3BttVivg+lS'
   }
  }
}
```

Figure 4: Example AS response (EDHOC+OSCORE, asymmetric).

## B.2. Using Symmetric Keys

In the case of a symmetric key, the AS MUST communicate the key to the client in the 'cnf' parameter of the access token response, as specified in Section 5.6.2. of [I-D.ietf-ace-oauth-authz]. AS MUST also select a key identifier, that MUST be included as the 'kid' parameter either directly in the 'cnf' structure, as in figure 4 of [I-D.ietf-ace-oauth-authz], or as the 'kid' parameter of the COSE_key, as in figure 6 of [I-D.ietf-ace-oauth-authz].

Figure 5 shows an example of the necessary parameters in the AS response to the access token request when EDHOC is used. The example uses CBOR diagnostic notation without tag and value abbreviations.

```
Header: Created (Code=2.01)
Content-Type: "application/cose+cbor"
Payload:
{
  "access_token" : b64'SlAV32hkKG ...
   (remainder of access token omitted for brevity)',
  "profile" : "coap_oscore_edhoc",
  "expires_in" : "3600",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'5tOS+h42dkw',
      "k" : b64'+a+Dg2jjU+eIiOFCa9lObw'
    }
  }
}
```

Figure 5: Example AS response (EDHOC+OSCORE, symmetric).

In both cases, the AS MUST also include the same key identifier as 'kid' parameter in the access token metadata. If the access token is a CWT [RFC8392], the key identifier MUST be placed inside the 'cnf' claim as 'kid' parameter of the COSE_Key or directly in the 'cnf' structure (if the key is only referenced).

Figure 6 shows an example CWT containing the necessary EDHOC+OSCORE parameters in the 'cnf' claim, in CBOR diagnostic notation without tag and value abbreviations.

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kty" : "Symmetric",
      "kid" : b64'5tOS+h42dkw',
      "k" : b64'+a+Dg2jjU+eIiOFCa9lObw'
  }
}
```

Figure 6: Example CWT with EDHOC+OSCORE, symmetric case.

All other parameters defining OSCORE security context are derived from EDHOC message exchange, including the master secret (see Appendix C.2 of [I-D.selander-ace-cose-ecdhe]).

### B.3. Processing

To provide forward secrecy and mutual authentication in the case of pre-shared keys, pre-established raw public keys or with X.509 certificates it is RECOMMENDED to use EDHOC [I-D.selander-ace-cose-ecdhe] to generate the keying material. EDHOC MUST be used as defined in Appendix C of [I-D.selander-ace-cose-ecdhe], with the following additions and modifications.

The first EDHOC message is sent after the access token is posted to the /authz-info resource of the RS as specified in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. Then the EDHOC message_1 is sent and the EDHOC protocol is initiated [I-D.selander-ace-cose-ecdhe]).

Before the RS continues with the EDHOC protocol and responds to this token submission request, additional verifications on the access token are done: the RS SHALL process the access token according to [I-D.ietf-ace-oauth-authz]. If the token is valid then the RS continues processing EDHOC following Appendix C of [I-D.selander-ace-cose-ecdhe], otherwise it discontinues EDHOC and responds with the error code as specified in [I-D.ietf-ace-oauth-authz].

- In case the EDHOC verification fails, the RS MUST return an error response to the client with code 4.01 (Unauthorized).
- If RS has an access token for C but not for the resource that C has requested, RS MUST reject the request with a 4.03 (Forbidden).
- If RS has an access token for C but it does not cover the action C requested on the resource, RS MUST reject the request with a 4.05 (Method Not Allowed).
- If all verifications above succeeds, further communication between client and RS is protected with OSCORE, including the RS response to the OSCORE request.

In the case of EDHOC being used with symmetric keys, the protocol in Section 5 of [I-D.selander-ace-cose-ecdhe] MUST be used. If the key is asymmetric, the RS MUST also use an asymmetric key for authentication. This key is known to the client through the access token response (see Section 5.6.2 of [I-D.ietf-ace-oauth-authz]). In this case the protocol in Section 4 of [I-D.selander-ace-cose-ecdhe] MUST be used.

Figure 7 illustrates the message exchanges for using OSCORE+EDHOC (step C in figure 1 of [I-D.ietf-ace-oauth-authz]).

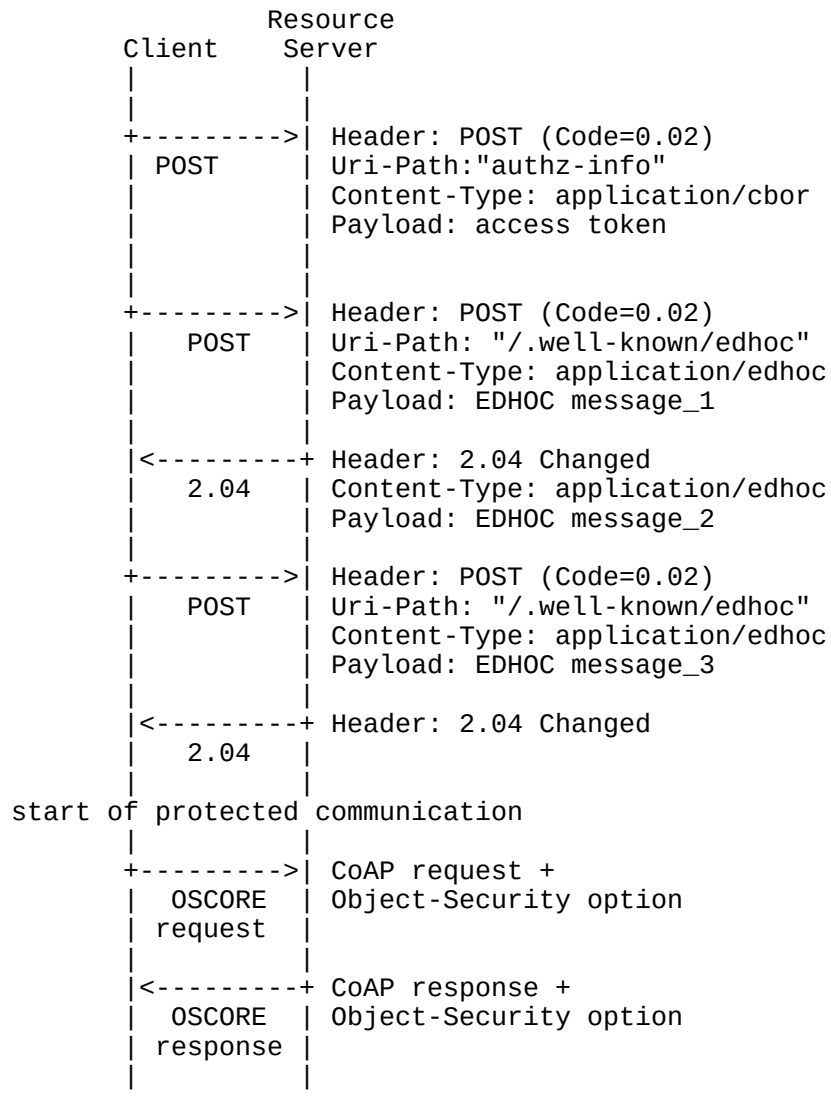



Figure 7: Access token and key establishment with EDHOC

Acknowledgments

The authors wish to thank Jim Schaad, Goeran Selander and Marco Tiloca for the input on this memo. The error responses specified in Appendix B.3 were originally specified by Gerdes et al. in [I-D.gerdes-ace-dcaf-authorize].

Authors' Addresses

Ludwig Seitz
RISE SICS AB
Scheelevagen 17
Lund 22370
Sweden

Email: ludwig.seitz@ri.se

Francesca Palombini
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Martin Gunnarsson
RISE SICS AB
Scheelevagen 17
Lund 22370
Sweden

Email: martin.gunnarsson@ri.se

Goeran Selander
Ericsson AB
Farogatan 6
Kista SE-16480 Stockholm
Sweden

Email: goran.selander@ericsson.com

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.

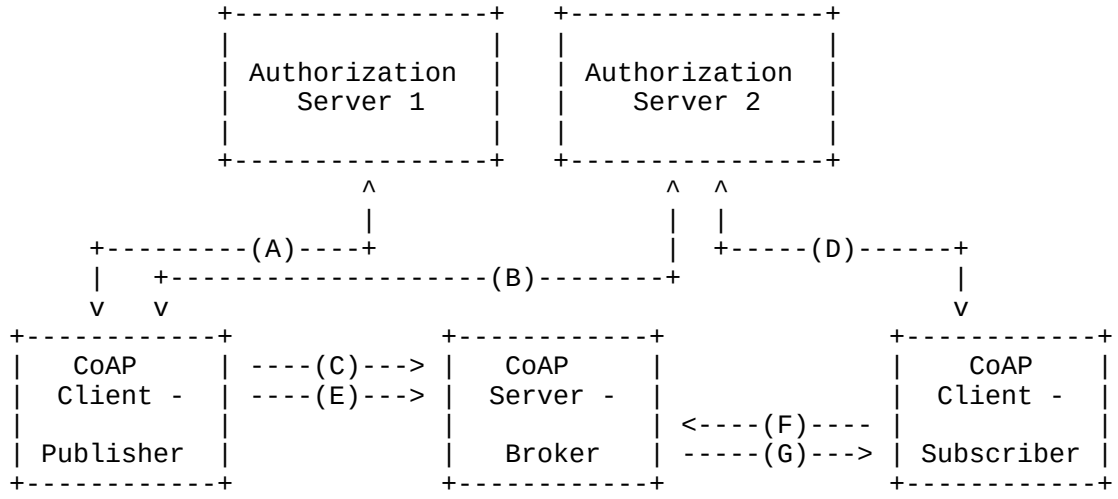



Figure 1: Architecture CoAP pubsub with Authorization Servers

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.

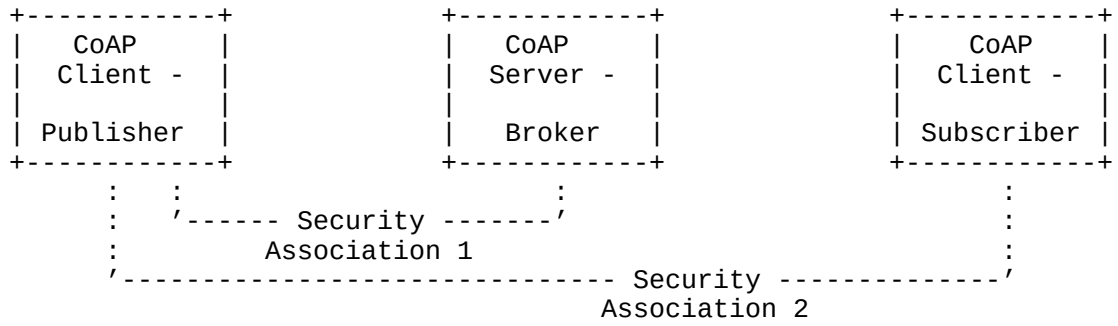
2. The Publisher requests access to a specific topic at the AS2

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.

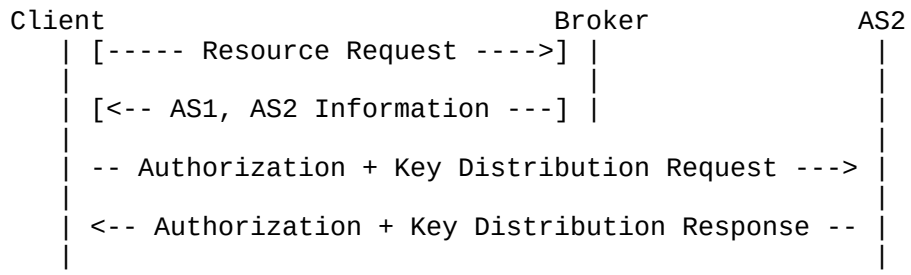



Figure 2: B: Access request - response

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.

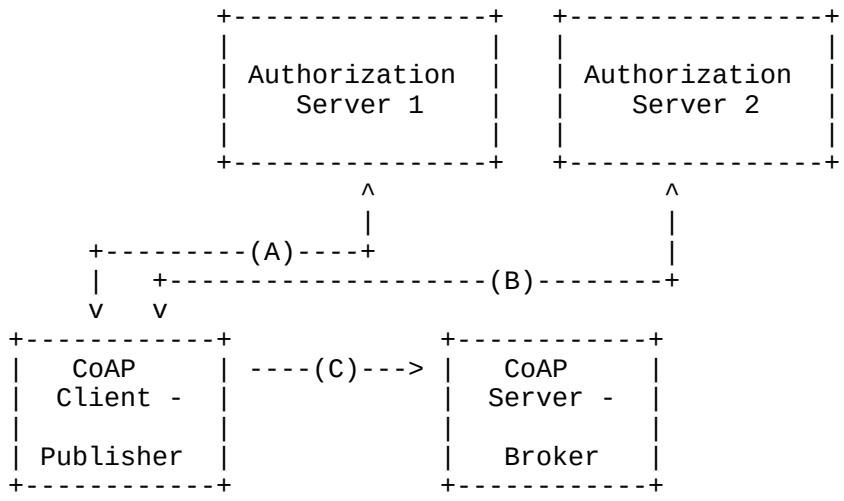



Figure 4: Phase 1: Publisher side

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.

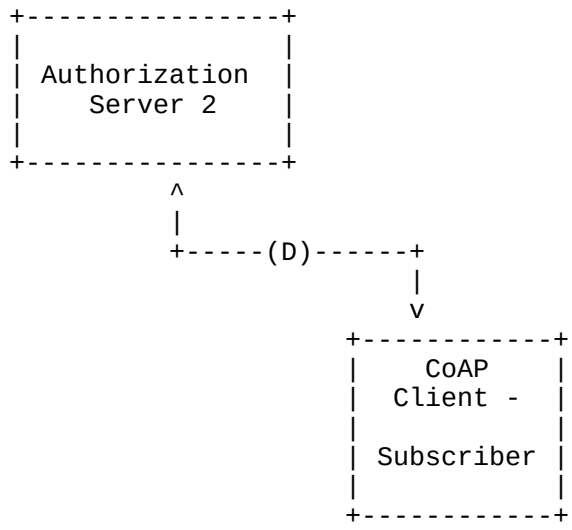



Figure 7: Phase 2: Subscriber side

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.

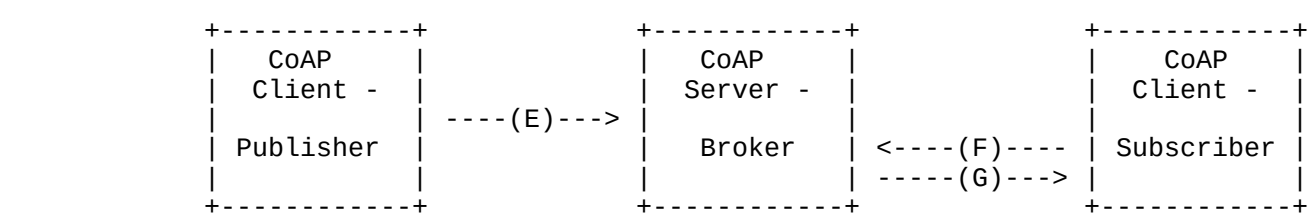



Figure 10: Phase 3: Secure communication between Publisher and Subscriber

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.

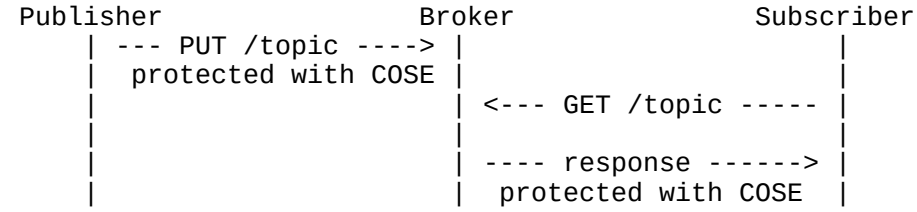



Figure 11: (E), (F), (G): Example of protected communication

## 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: December 29, 2018

F. Palombini
Ericsson AB
M. Tiloca
RISE SICS

June 27, 2018

# Key Provisioning for Group Communication using ACE
# draft-palombini-ace-key-groupcomm-01

## Abstract

This document defines a message format for distributing keying material in group communication scenarios (such as based on multicast or publisher-subscriber model) using the ACE framework.

## 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

This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to define the format of messages used to distribute the keying material in a group communication scenario. Profiles that use group communication can build on this document to specify exactly which of the message parameters defined in this documents are used, and what are 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.

### 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].

## 2. Overview

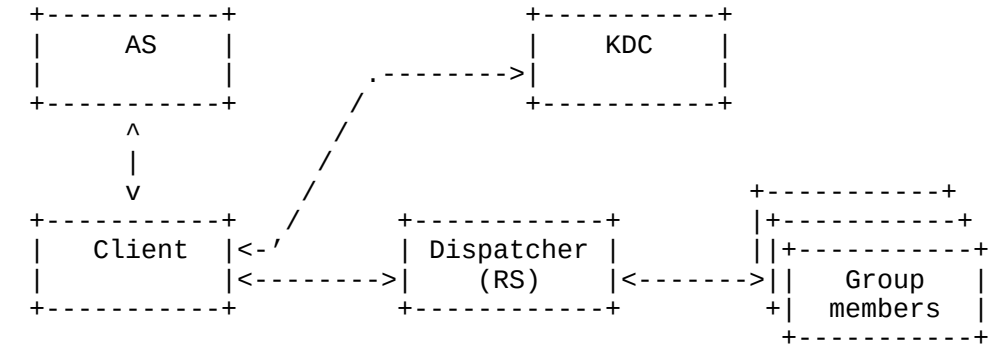



Figure 1: Key Distribution Participants

Participants:

- Client: Node that wants to join the group communication. It can either want write rights or read rights.

- AS: Same as AS in the ACE Framework; it contains policies, and knows if a node is allowed to join the group with write or read rights.

- Key Distribution Center: 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, it corresponds to the RS in the ACE Framework.

- Dispatcher: this is the entity the Client wants to securely communicate with and is responsible for distribution of group messages. It can be an existing node, such as the Broker in a pub-sub setting (in which case the Dispatcher is also a RS), or it can be implicit, as in the multicast communication setting, where the message distribution is done by transmitting to a multicast IP address, and entrusting message delivery to the transport channel.

This document specifies the message flows and formats for adding a node to a group, as well as for the distribution of keying material to joining nodes. Also, it briefly mentions the node's removal from a group and the consequent rekeying process.

The high level overview of the message flow for a node joining a group communication setting is shown in Figure 2.

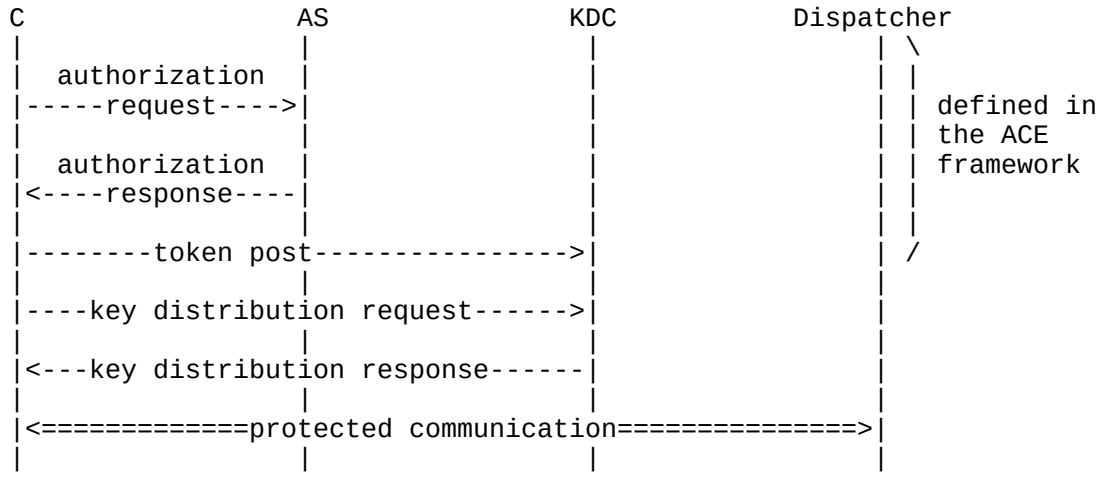



Figure 2: Key Distribution Message Flow

## 3. Addition to the Group

This section describes in detail the message formats exchanged by the participants when a node requests access to the group. The first part of the exchange is based on ACE [I-D.ietf-ace-oauth-authz], where the KDC takes the role of RS.

### 3.1. Authorization Request

The Authorization Request sent from the Client to the AS (as defined in [I-D.ietf-ace-oauth-authz], Section 5.6.1, 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, as well as the role the Client wishes to take, if necessary. 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

    How exactly the group or topic identifier and the roles are encoded is application specific.

- aud, with value an identifier of the KDC.

- cnf, containing the public key (or 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 (as defined in [I-D.ietf-ace-oauth-authz], Section 5.6.2, MUST contain the following parameters:

- access_token, containing all the parameters defined below (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.

- cnf if symmetric keys are used, not present if asymmetric keys are used, contains the symmetric pop key that the Client is supposed to use with the KDC.

- rs_cnf if asymmetric keys are used, contains information about the public key of the KDC. Not present if symmetric keys are used.

- 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 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.

How exactly the group or topic identifier and the roles are encoded is application specific.

- Other additional parameters as defined in [I-D.ietf-ace-oauth-authz], if necessary.

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 profile defines it, the Client MAY use a different endpoint 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.

The same types of messages 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 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.

### 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 (which is associated in the KDC 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 or topic identifier and role(s) that the Client is authorized to access, 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 supplied 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 3.

| Name | Label | CBOR Type | Value Registry | Description |
|---|---|---|---|---|
| exp | TBD | Integer or floating-point number | COSE Key Common Parameters | Expiration time in seconds |

Figure 3: COSE Key Common Header Parameter 'exp'

Additionally, 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 revocation and renewal of group keying (rekeying) performed by the KDC. The exact format and content depend on the specific rekeying algorithm 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.

TBD: define for verification failure

## 5. Remove a Node from the Group

This section describes at a high level how a node can be removed from the group.

### 5.1. Not authorized anymore

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 generate and distribute the new keying material to all authorized members of the group, as well as to remove the unauthorized node from the list of members (if the KDC keeps track of that). The KDC relies on the specific rekeying algorithm used in the group, such as e.g. [RFC2093], [RFC2094] or [RFC2627], and the related management key material.

### 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 to the KDC to exit the group. The KDC can then generate and distribute the new keying material to all authorized members of the group, as well as remove the leaving node from the list of members (if the KDC keeps track of that).

Note that, as long as the node is authorized to join the group, i.e. it has a valid Access Token, it can re-request to join the group directly to the KDC without needing to retrieve a new Access Token. 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 participate to.

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 MAY 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.

In some cases, it is not necessary to include the scope parameter, for instance if the KDC maintains a list of active group members for each managed group, and the Client is member of only one group. The Client MUST include the scope parameter if it is a member of multiple groups under the same KDC.

### 6.2. Key Re-Distribution Response

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.

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 field:

- get_pub_keys, which has for 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.

Additionally, this request MAY contain the following parameter, which, if included, MUST have the corresponding value:

- 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.

  In some cases, it is not necessary to include the scope parameter, for instance if the KDC maintains a list of active group members for each managed group, and if the specified identifiers allow to retrieve public keys with no ambiguity. The Client MUST include the scope parameter if it is a member of multiple groups under the same KDC.

If the KDC can not unambiguously identify the nodes specified in the 'get_pub_keys' parameter, it MUST reply with an error message. In this case, the Client can issue a new Public Key Request specifying the group in the 'scope' parameter.

TODO: define error

### 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 algorithm 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-12 (work in progress), May 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.

[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-04 (work in progress), March 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 and Goeran Selander.

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 SICS
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2018

M. Tiloca
RISE SICS
J. Park
Universitaet Duisburg-Essen
June 29, 2018

# Joining OSCORE groups in ACE
# draft-tiloca-ace-oscoap-joining-04

## Abstract

This document describes a method to join a group where communications are based on CoAP and secured with Object Security for Constrained RESTful Environments (OSCORE). The proposed method delegates the authentication and authorization of client nodes that join an OSCORE group through a Group Manager server. This approach builds on the ACE framework for Authentication and Authorization, and leverages protocol-specific profiles of ACE to achieve communication security, proof-of-possession and server authentication.

Table of Contents



## 1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] is a method for application-layer protection of the Constrained Application Protocol (CoAP) [RFC7252], using CBOR Object Signing and Encryption (COSE) [RFC8152] and enabling end-to-end security of CoAP payload and options.

As described in [I-D.ietf-core-oscore-groupcomm], OSCORE may be used also to protect CoAP group communication over IP multicast [RFC7390]. This relies on a Group Manager entity, which is responsible for managing an OSCORE group, where members exchange CoAP messages secured with OSCORE. In particular, the Group Manager coordinates the join process of new group members and can be responsible for multiple groups.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines how a client joins an OSCORE group through a resource server acting as Group Manager. The client acting as joining node relies on an Access Token, which is bound to a proof-of-possession key and authorizes the access to a specific join resource at the Group Manager. Messages exchanged among the participants follow the formats defined in

[I-D.palombini-ace-key-groupcomm] for provisioning keying material in group communication scenarios.

In order to achieve communication security, proof-of-possession and server authentication, the client and the Group Manager leverage protocol-specific profiles of ACE. These include 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].

## 2. Protocol Overview

Group communication for CoAP over IP multicast has been enabled in [RFC7390] and can be secured with Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] as described in [I-D.ietf-core-oscore-groupcomm]. A network node explicitly joins an OSCORE group, by interacting with the responsible Group Manager. Once registered in the group, the new node can securely exchange messages with other group members.

This specification describes how a network node joins an OSCORE group by using the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- The Group Manager acts as Resource Server (RS), and hosts one join resource for each OSCORE group it manages. Each join resource is exported by a distinct join endpoint. During the join process, the Group Manager provides joining nodes with the parameters and keying material for taking part to secure communications in the group.

- The joining node acts as Client (C), and requests to join an OSCORE group by accessing the related join endpoint at the Group Manager.

- The Authorization Server (AS) 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 by means of DTLS [RFC6347] or OSCORE [I-D.ietf-core-object-security]. Further details on how the AS secures communications (with the joining node and the Group Manager) depend on the specifically used profile of ACE, and are out of the scope of this specification.

The following steps are performed for joining an OSCORE group. Messages exchanged among the participants follow the formats defined in [I-D.palombini-ace-key-groupcomm], and are further specified in Section 3 and Section 4 of this document. The Group Manager acts as the Key Distribution Center (KDC) referred in [I-D.palombini-ace-key-groupcomm].

1. The joining node requests an Access Token from the AS, 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 alternatives to provide a secure communication channel include 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.

## 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 [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 2 of [I-D.ietf-ace-dtls-authorize] 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 includes:
  - in the first element, the Group Identifier (Gid) of the group to join under the Group Manager. The value of this identifier may not fully coincide with the Gid value currently associated to the group, e.g. if the Gid is composed of a variable part

such as a Group Epoch (see Appendix C of [I-D.ietf-core-oscore-groupcomm]).

* in the second element, which MUST be present, the role(s) that the joining node intends to have in the group it intends to join. Possible values are: "requester"; "listener"; and "pure listener". Possible combinations are: "requester and listener"; and "requester and pure listener". Multiple roles are specified in the form of a CBOR array.

o The '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 5.6.4.4 of [I-D.ietf-ace-oauth-authz]).

In particular, if symmetric keys are used, the AS generates a proof-of-possession key, binds it to the Access Token, and provides it to the joining node in the 'cnf' parameter of the Access Token response. Instead, if asymmetric keys are used, the joining node provides its own public key to the AS in the 'cnf' parameter of the Access Token request. Then, the AS uses it as proof-of-possession key bound to

the Access Token, and provides the joining node with the Group Manager's public key in the 'rs_cnf' parameter of the Access Token response.

# 4. Joining Node to Group Manager

First, the joining node posts the Access Token to the /authz-info endpoint at the Group Manager, in accordance with the Token post defined in Section 3.3 of [I-D.palombini-ace-key-groupcomm]. Then, the joining node establishes a secure channel with the Group Manager, according to what 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 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 Group Manager is configured to store the public keys of the group members and, at the same time, the joining node wants to retrieve such public keys during the joining process (see Section 5). In any other case, this parameter MUST NOT be present.

- The 'client_cred' parameter, if present, includes the public key or certificate of the joining node. Specifically, it includes the public key of the joining node if the Group Manager is configured to store the public keys of the group members, or the certificate of the joining node otherwise. This parameter MAY be omitted if: i) public keys are used as proof-of-possession keys between the joining node and the Group Manager; or ii) the joining node is asking to access the group exclusively as pure listener; or iii) the Group Manager already acquired this information during a previous join process. In any other case, this parameter MUST be present.

- The 'pub_keys_repos' parameter MAY be present if the 'client_cred' parameter is both present and with value a certificate of the joining node. If present, this parameter contains the list of public key repositories storing the certificate of the joining node. In any other case, this parameter MUST NOT be present.

### 4.2. Join Response

The Group Manager processes the request according to [I-D.ietf-ace-oauth-authz]. If this yields a positive outcome, the Group Manager updates the group membership by registering the joining node as a new member of the OSCORE group.

Then, the Group Manager replies to the joining node providing the information necessary to participate in the group communication. This join response follows the format of the Key Distribution success Response message defined in Section 4.2 of [I-D.palombini-ace-key-groupcomm]. In particular:

- The 'key' parameter includes what the joining node needs in order to set up the OSCORE Security Context as 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 Master Secret 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 Endpoint 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) currently associated to the group.
  - The 'kdf' parameter, if present, has as value the KDF algorithm used in the group.
  - The 'slt' parameter, if present, has as value the OSCORE Master Salt.

  * The 'cs_alg' parameter MUST be present and has as value the countersignature algorithm used in the group.

- The 'pub_keys' parameter is present only if the 'get_pub_keys' parameter was present in the join request. If present, this parameter includes the public keys of the group members that are relevant to the joining node. That is, it includes: i) the public keys of the non-pure listeners currently in the group, in case the joining node is configured (also) as 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]), as well as the rekeying protocol used to renew the keying material in the group (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]).

- The 'mgt_key_material' parameter SHOULD be present and includes the administrative keying material that the joining node requires to participate in the rekeying process led by the Group Manager. The exact content and format depend on the specific rekeying protocol used in the group.

Finally, the joining node uses the information received in the join response to set up the OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. From then on, the joining node can exchange group messages secured with OSCORE as described in Section 4 of [I-D.ietf-core-oscore-groupcomm].

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. A possible approach for the node to renew the OSCORE Security Context through the Group Manager is described in Section 6 of [I-D.palombini-ace-key-groupcomm].

## 5. Public Keys of Joining Nodes

Source authentication of OSCORE messages exchanged within the group is ensured by means of digital counter signatures [I-D.ietf-core-oscore-groupcomm]. Therefore, group members must be able to retrieve each other's public key from a trusted key repository, in order to verify the source authenticity of incoming group messages.

Upon joining an OSCORE group, a joining node is expected to make its own public key available to the other group members, either through the Group Manager or through another trusted, publicly available, key repository. However, this is not required for a node that joins a group exclusively as pure listener.

As also discussed in Section 6 of [I-D.ietf-core-oscore-groupcomm], it is recommended that the Group Manager is configured to store the public keys of the group members and to provide them upon request. If so, three cases can occur when a new node joins a group.

- The Group Manager already acquired the public key of the joining node during a previous join process. In this case, the joining node 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. A possible approach to do this through the Group Manager is described in Section 7 of [I-D.palombini-ace-key-groupcomm].

On the other hand, in case the Group Manager is not configured to store public keys of group members, the joining node provides the Group Manager with its own certificate in the 'client_cred' parameter of the join request targeting the join endpoint (see Section 4.1). Then, the Group Manager validates and handles the certificate, for instance as described in Appendix D.2 of [I-D.ietf-core-oscore-groupcomm].

## 6. Security Considerations

The method described in this document leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

- Management of group keying material (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]). This includes the need to revoke and renew the keying material currently used in the OSCORE group, upon changes in the group membership. In particular, renewing the keying material is required upon a new node joining the group, in order to preserve backward security. That is, the Group Manager should renew the keying material before completing the join process and sending a join response. Such a join response provides the joining node with the updated keying material just established in the group. The Group Manager is responsible to enforce rekeying policies and accordingly update the keying material in the groups of its competence (see Section 6 of [I-D.ietf-core-oscore-groupcomm]).

- 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.

- Provisioning and retrieval of public keys (see Appendix D.2 of [I-D.ietf-core-oscore-groupcomm]). This provides guidelines about how to ensure the availability of group members' public keys, possibly relying on the Group Manager as trusted key repository (see Section 6 of [I-D.ietf-core-oscore-groupcomm]).

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 the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], as well as from the specific profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

## 7. IANA Considerations

This document has no actions for IANA.

## 8. Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Martin Gunnarsson, Francesca Palombini, Jim Schaad, Ludwig Seitz, 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.

## 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-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-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-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-01 (work in progress), June 2018.

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

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

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

## 9.2. Informative References

[I-D.ietf-ace-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.

[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>.

## Authors' Addresses

Marco Tiloca
RISE SICS
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