Stream: Internet Engineering Task Force (IETF)
RFC: 9202
Category: Standards Track
Published: August 2022
ISSN: 2070-1721
Authors: S. Gerdes, *Universität Bremen TZI*; O. Bergmann, *Universität Bremen TZI*; C. Bormann, *Universität Bremen TZI*; G. Selander, *Ericsson AB*; L. Seitz, *Combitech*

RFC 9202
Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)

Abstract

This specification defines a profile of the Authentication and Authorization for Constrained Environments (ACE) framework that allows constrained servers to delegate client authentication and authorization. The protocol relies on DTLS version 1.2 or later for communication security between entities in a constrained network using either raw public keys or pre-shared keys. A resource-constrained server can use this protocol to delegate management of authorization information to a trusted host with less-severe limitations regarding processing power and memory.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9202.

Copyright Notice

Table of Contents

1. Introduction

This specification defines a profile of the ACE framework [RFC9200]. In this profile, a client (C) and a resource server (RS) use the Constrained Application Protocol (CoAP) [RFC7252] over DTLS version 1.2 [RFC6347] to communicate. This specification uses DTLS 1.2 terminology, but later versions such as DTLS 1.3 [RFC9147] can be used instead. The client obtains an access token bound to a key (the proof-of-possession (PoP) key) from an authorization server (AS) to prove its authorization to access protected resources hosted by the resource server. Also, the client and the resource server are provided by the authorization server with the necessary keying material to establish a DTLS session. The communication between the client and authorization server may also be secured with DTLS. This specification supports DTLS with raw public keys (RPKs) [RFC7250] and with pre-shared keys (PSKs) [RFC4279]. How token introspection [RFC7662] is performed between the RS and AS is out of scope for this specification.

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

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

The resource server needs access to the token in order to complete this exchange. For the RPK mode, the client must upload the access token to the resource server before initiating the handshake, as described in Section 5.10.1 of the ACE framework [RFC9200].

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

As recommended in Section 5.8 of [RFC9200], this specification uses Concise Binary Object Representation (CBOR) web tokens to convey claims within an access token issued by the server. While other formats could be used as well, those are out of scope for this document.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in [RFC9200] and [RFC9201].

The authorization information (authz-info) resource refers to the authorization information endpoint, as specified in [RFC9200]. The term `claim` is used in this document with the same semantics as in [RFC9200], i.e., it denotes information carried in the access token or returned from introspection.

Throughout this document, examples for CBOR data items are expressed in CBOR extended diagnostic notation as defined in Section 8 of [RFC8949] and Appendix G of [RFC8610] ("diagnostic notation"), unless noted otherwise. We often use diagnostic notation comments to provide a textual representation of the numeric parameter names and values.

2. Protocol Overview

The CoAP-DTLS profile for ACE specifies the transfer of authentication information and, if necessary, authorization information between the client (C) and the resource server (RS) during setup of a DTLS session for CoAP messaging. It also specifies how the client can use CoAP over DTLS to retrieve an access token from the authorization server (AS) for a protected resource hosted on the resource server. As specified in Section 6.7 of [RFC9200], use of DTLS for one or both of these interactions is completely independent.

This profile requires the client to retrieve an access token for the protected resource(s) it wants to access on the resource server, as specified in [RFC9200]. Figure 1 shows the typical message flow in this scenario (messages in square brackets are optional):

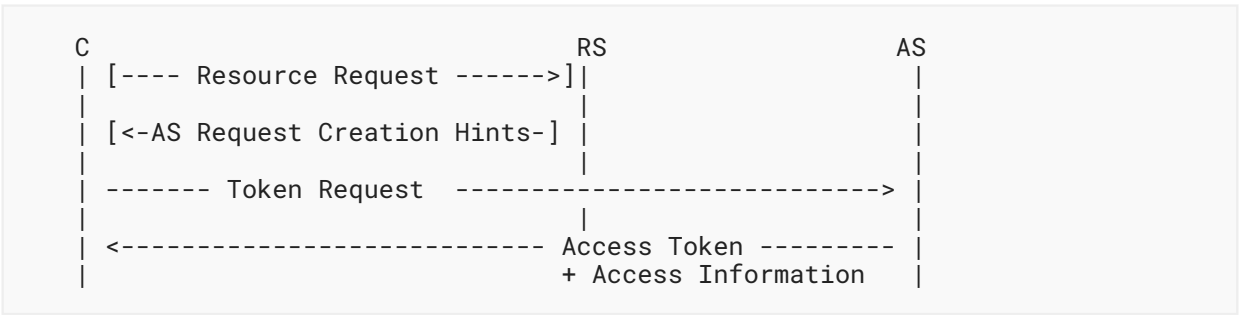

Figure 1: Retrieving an Access Token

To determine the authorization server in charge of a resource hosted at the resource server, the client can send an initial Unauthorized Resource Request message to the resource server. The resource server then denies the request and sends an AS Request Creation Hints message containing the address of its authorization server back to the client, as specified in Section 5.3 of [RFC9200].

Once the client knows the authorization server's address, it can send an access token request to the token endpoint at the authorization server, as specified in [RFC9200]. As the access token request and the response may contain confidential data, the communication between the client and the authorization server must be confidentiality protected and ensure authenticity. The client is expected to have been registered at the authorization server, as outlined in Section 4 of [RFC9200].

The access token returned by the authorization server can then be used by the client to establish a new DTLS session with the resource server. When the client intends to use an asymmetric proof-of-possession key in the DTLS handshake with the resource server, the client **MUST** upload the access token to the authz-info resource, i.e., the authz-info endpoint, on the resource server before starting the DTLS handshake, as described in Section 5.10.1 of [RFC9200]. In case the client uses a symmetric proof-of-possession key in the DTLS handshake, the procedure above **MAY** be used, or alternatively the access token **MAY** instead be transferred in the DTLS ClientKeyExchange message (see Section 3.3.2). In any case, DTLS **MUST** be used in a mode that provides replay protection.

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

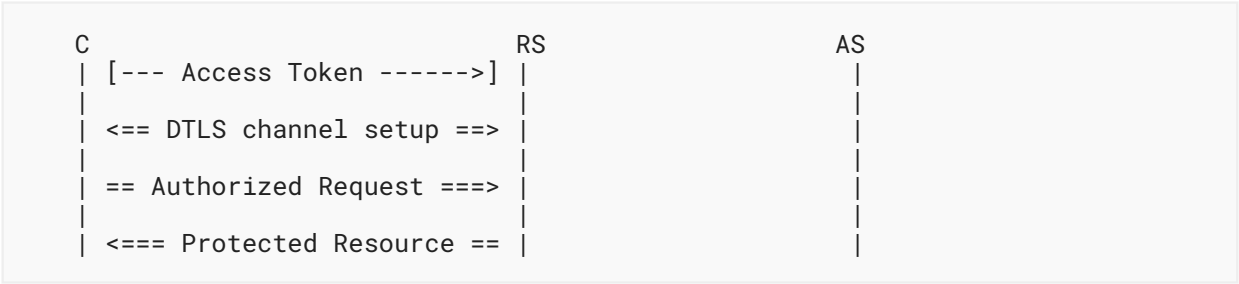

Figure 2: Protocol Overview

3. Protocol Flow

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

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

3.1. Communication between the Client and the Authorization Server

To retrieve an access token for the resource that the client wants to access, the client requests an access token from the authorization server. Before the client can request the access token, the client and the authorization server **MUST** establish a secure communication channel. This profile assumes that the keying material to secure this communication channel has securely been obtained either by manual configuration or in an automated provisioning process. The following requirements, in alignment with Section 6.5 of [RFC9200], therefore must be met:

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

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

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

3.2. Raw Public Key Mode

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

3.2.1. Access Token Retrieval from the Authorization Server

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

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

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  / grant_type / 33 : / client_credentials / 2,
  / audience /    5 : "tempSensor4711",
  / req_cnf /     4 : {
    / COSE_Key / 1 : {
      / kty /  1 : / EC2 /   2,
      / crv / -1 : / P-256 / 1,
      / x /   -2 : h'e866c35f4c3c81bb96a1/.../',
      / y /   -3 : h'2e25556be097c8778a20/.../'
    }
  }
}
```

Figure 3: Access Token Request Example for RPK Mode

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

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

The response **MUST** contain an `ace_profile` parameter if the `ace_profile` parameter in the request is empty and **MAY** contain this parameter otherwise (see Section 5.8.2 of [RFC9200]). This parameter is set to `coap_dtls` to indicate that this profile **MUST** be used for communication between the client and the resource server. The response also contains an access token with information for the resource server about the client's public key. The authorization server **MUST** return in its response the parameter `rs_cnf` unless it is certain that the client already knows the public key of the resource server. The authorization server **MUST** ascertain that the RPK specified in `rs_cnf` belongs to the resource server that the client wants to communicate with. The authorization server **MUST** protect the integrity of the access token such that the resource server can detect unauthorized changes. If the access token contains confidential data, the authorization server **MUST** also protect the confidentiality of the access token.

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

An example access token response from the authorization server to the client is depicted in Figure 4. Here, the contents of the `access_token` claim have been truncated to improve readability. For the client, the response comprises Access Information that contains the server's

public key in the `rs_cnf` parameter. Caching proxies process the Max-Age option in the CoAP response, which has a default value of 60 seconds (Section 5.6.1 of [RFC7252]). The authorization server **SHOULD** adjust the Max-Age option such that it does not exceed the `expires_in` parameter to avoid stale responses.

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 3560
Payload:
{
  / access_token / 1 : b64'SlAV32hk'/...
   (remainder of CWT omitted for brevity;
   CWT contains the client's RPK in the cnf claim)/,
  / expires_in /  2 : 3600,
  / rs_cnf /     41 : {
    / COSE_Key /  1 : {
      / kty /  1 : / EC2 /   2,
      / crv / -1 : / P-256 / 1,
      / x /   -2 : h'd7cc072de2205bdc1537/.../',
      / y /   -3 : h'f95e1d4b851a2cc80fff/.../'
    }
  }
}
```

Figure 4: Access Token Response Example for RPK Mode

3.2.2. DTLS Channel Setup between the Client and Resource Server

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

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

To be consistent with [RFC7252], which allows for shortened Message Authentication Code (MAC) tags in constrained environments, an implementation that supports the RPK mode of this profile **MUST** at least support the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. As discussed in [RFC7748], new Elliptic Curve Cryptography (ECC) curves have been defined recently that are considered superior to the so-called NIST curves. Implementations of this profile **MUST**

therefore implement support for curve25519 (cf. [RFC8032], [RFC8422]), as this curve is said to be efficient and less dangerous, regarding implementation errors, than the secp256r1 curve mandated in [RFC7252].

The resource server **MUST** check if the access token is still valid, if the resource server is the intended destination (i.e., the audience) of the token, and if the token was issued by an authorized authorization server (see also Section 5.10.1.1 of [RFC9200]). The access token is constructed by the authorization server such that the resource server can associate the access token with the client's public key. The `cnf` claim **MUST** contain either the client's RPK or, if the key is already known by the resource server (e.g., from previous communication), a reference to this key. If the authorization server has no certain knowledge that the client's key is already known to the resource server, the client's public key **MUST** be included in the access token's `cnf` parameter. If CBOR web tokens [RFC8392] are used (as recommended in [RFC9200]), keys **MUST** be encoded as specified in [RFC8747]. A resource server **MUST** have the capacity to store one access token for every proof-of-possession key of every authorized client.

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

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

3.3. Pre-shared Key Mode

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

3.3.1. Access Token Retrieval from the Authorization Server

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

The authorization server **MUST** determine the authorization rules for the client it communicates with, as defined by the resource owner, and generate the access token accordingly. If the authorization server authorizes the client, it returns an AS-to-client response. If the `ace_profile` parameter is present, it is set to `coap_dtls`. The authorization server **MUST** ascertain that the access token is generated for the resource server that the client wants to communicate with. Also, the authorization server **MUST** protect the integrity of the access token to ensure that the resource server can detect unauthorized changes. If the token contains confidential data, such as

the symmetric key, the confidentiality of the token **MUST** also be protected. Depending on the requested token type and algorithm in the access token request, the authorization server adds Access Information to the response that provides the client with sufficient information to set up a DTLS channel with the resource server. The authorization server adds a `cnf` parameter to the Access Information carrying a `COSE_Key` object that informs the client about the shared secret that is to be used between the client and the resource server. To convey the same secret to the resource server, the authorization server can include it directly in the access token by means of the `cnf` claim or provide sufficient information to enable the resource server to derive the shared secret from the access token. As an alternative, the resource server **MAY** use token introspection to retrieve the keying material for this access token directly from the authorization server.

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

```
POST coaps://as.example.com/token
Content-Format: application/ace+cbor
Payload:
{
  / audience / 5 : "smokeSensor1807"
}
```

Figure 5: Example Access Token Request, (Implicit) Symmetric PoP Key

A corresponding example access token response is illustrated in Figure 6. In this example, the authorization server returns a 2.01 response containing a new access token (truncated to improve readability) and information for the client, including the symmetric key in the `cnf` claim. The information is transferred as a CBOR data structure as specified in [RFC9200].

```
2.01 Created
Content-Format: application/ace+cbor
Max-Age: 85800
Payload:
{
   / access_token /  1 : h'd08343a1/...
     (remainder of CWT omitted for brevity)/',
   / token_type /   34 : / PoP / 2,
   / expires_in /    2 : 86400,
   / ace_profile /  38 : / coap_dtls / 1,
   / cnf /           8 : {
     / COSE_Key / 1 : {
       / kty / 1 : / symmetric / 4,
       / kid / 2 : h'3d027833fc6267ce',
       / k /  -1 : h'73657373696f6e6b6579'
     }
   }
}
```

Figure 6: Example Access Token Response, Symmetric PoP Key

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

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

```
/ cnf / 8 : {
  / COSE_Key / 1 : {
    / kty / 1 : / symmetric / 4,
    / kid / 2 : h'3d027833fc6267ce'
  }
}
```

Figure 7: Access Token without Keying Material

A response that declines any operation on the requested resource is constructed according to Section 5.2 of [RFC6749] (cf. Section 5.8.3 of [RFC9200]). Figure 8 shows an example for a request that has been rejected due to invalid request parameters.

```
    4.00 Bad Request
    Content-Format: application/ace+cbor
    Payload:
    {
      / error / 30 : / invalid_request / 1
    }
```

Figure 8: Example Access Token Response with Reject

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

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

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

In order to generate a new symmetric key to be used by the client and resource server, the authorization server generates a new key identifier that **MUST** be unique among all key identifiers used by the authorization server for this resource server. The authorization server then uses the key derivation key shared with the resource server to derive the symmetric key, as specified below. Instead of providing the keying material in the access token, the authorization server includes the key identifier in the `kid` parameter (see Figure 7). This key identifier enables the resource server to calculate the symmetric key used for the communication with the client using the key derivation key and a key derivation function (KDF) to be defined by the application, for example, HKDF-SHA-256. The key identifier picked by the authorization server **MUST** be unique for each access token where a unique symmetric key is required.

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

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

where:

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

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

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

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

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

3.3.2. DTLS Channel Setup between the Client and Resource Server

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

The client checks if the payload of the access token response contains an `access_token` parameter and a `cnf` parameter. With this information, the client can initiate the establishment of a new DTLS channel with a resource server. To use DTLS with pre-shared keys, the client follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], using the key conveyed in the `cnf` parameter of the AS response as a PSK when constructing the premaster secret. To be consistent with the recommendations in [RFC7252], a client in the PSK mode **MUST** support the cipher suite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655].

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

```
{ / cnf / 8 : {
   / COSE_Key / 1 : {
      / kty / 1 : / symmetric / 4,
      / kid / 2 : h'3d027833fc6267ce'
    }
  }
}
```

Figure 9: Access Token Containing a Single `kid` Parameter

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

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

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

DTLS 1.3 [RFC9147] does not use the ClientKeyExchange message; for DTLS 1.3, the access token is placed in the `identity` field of a `PSKIdentity` within the `PreSharedKeyExtension` of the `ClientHello`.

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

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

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

When the resource server receives an access token, it **MUST** check if the access token is still valid, if the resource server is the intended destination (i.e., the audience of the token), and if the token was issued by an authorized authorization server. This specification implements access tokens as proof-of-possession tokens. Therefore, the access token is bound to a symmetric PoP key that is used as a shared secret between the client and the resource server. A resource server **MUST** have the capacity to store one access token for every proof-of-possession key of every authorized client. The resource server may use token introspection [RFC7662] on the access token to retrieve more information about the specific token. The use of introspection is out of scope for this specification.

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

3.4. Resource Access

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

With the successful establishment of the DTLS channel, the client and the resource server have proven that they can use their respective keying material. An access token that is bound to the client's keying material is associated with the channel. According to Section 5.10.1 of [RFC9200], there should be only one access token for each client. New access tokens issued by the authorization server **SHOULD** replace previously issued access tokens for the respective client. The resource server therefore needs a common understanding with the authorization server about how access tokens are ordered. The authorization server may, e.g., specify a `cti` claim for the access token (see Section 5.9.2 of [RFC9200]) to employ a strict order.

Any request that the resource server receives on a DTLS channel that is tied to an access token via its keying material **MUST** be checked against the authorization rules that can be determined with the access token. The resource server **MUST** check for every request if the access token is still valid. If the token has expired, the resource server **MUST** remove it. Incoming CoAP requests that are not authorized with respect to any access token that is associated with the client **MUST** be rejected by the resource server with a 4.01 response. The response **SHOULD** include AS Request Creation Hints, as described in Section 5.2 of [RFC9200].

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

1. The message was received on a secure channel that has been established using the procedure defined in this document.
2. The authorization information tied to the sending client is valid.
3. The request is destined for the resource server.
4. The resource URI specified in the request is covered by the authorization information.
5. The request method is an authorized action on the resource with respect to the authorization information.

Incoming CoAP requests received on a secure DTLS channel that are not thus authorized **MUST** be rejected according to Section 5.10.2 of [RFC9200]:

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 is covered by the authorization information but not the requested action.

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

If the client gets an error response containing AS Request Creation Hints (cf. Section 5.3 of [RFC9200]) as a response to its requests, it **SHOULD** request a new access token from the authorization server in order to continue communication with the resource server.

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

4. Dynamic Update of Authorization Information

Resource servers must only use a new access token to update the authorization information for a DTLS session if the keying material that is bound to the token is the same that was used in the DTLS handshake. By associating the access tokens with the identifier of an existing DTLS session,

the authorization information can be updated without changing the cryptographic keys for the DTLS communication between the client and the resource server, i.e., an existing session can be used with updated permissions.

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

Figure 10 depicts the message flow where the client requests a new access token after a security association between the client and the resource server has been established using this protocol. If the client wants to update the authorization information, the token request **MUST** specify the key identifier of the proof-of-possession key used for the existing DTLS channel between the client and the resource server in the `kid` parameter of the client-to-AS request. The authorization server **MUST** verify that the specified `kid` denotes a valid verifier for a proof-of-possession token that has previously been issued to the requesting client. Otherwise, the client-to-AS request **MUST** be declined with the error code `unsupported_pop_key`, as defined in Section 5.8.3 of [RFC9200].

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

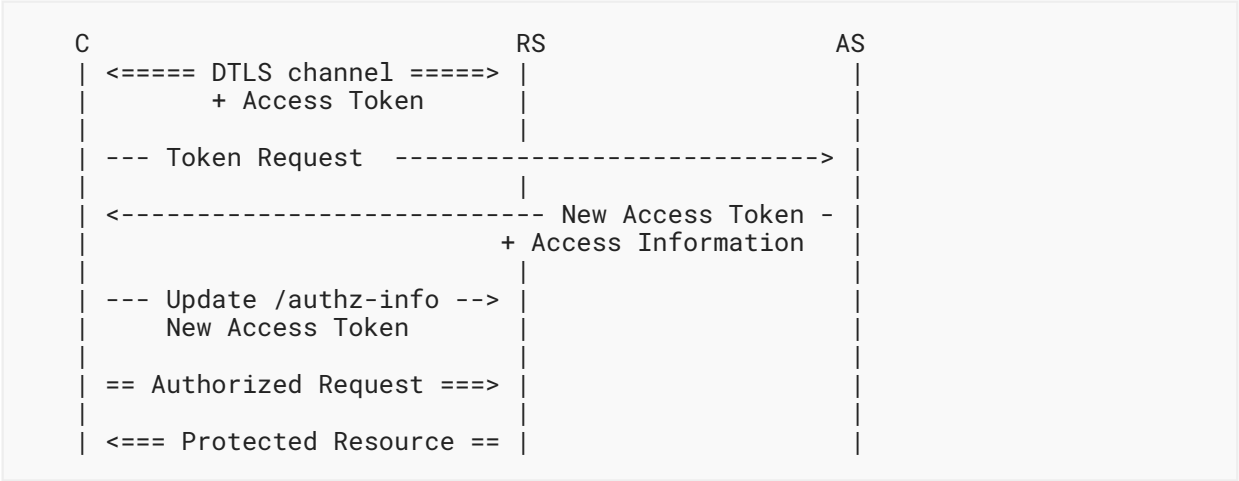

Figure 10: Overview of Dynamic Update Operation

5. Token Expiration

The resource server **MUST** delete access tokens that are no longer valid. DTLS associations that have been set up in accordance with this profile are always tied to specific tokens (which may be exchanged with a dynamic update, as described in Section 4). As tokens may become invalid at

any time (e.g., because they have expired), the association may become useless at some point. A resource server therefore **MUST** terminate existing DTLS association after the last access token associated with this association has expired.

As specified in Section 5.10.3 of [RFC9200], the resource server **MUST** notify the client with an error response with code 4.01 (Unauthorized) for any long-running request before terminating the association.

6. Secure Communication with an Authorization Server

As specified in the ACE framework (Sections 5.8 and 5.9 of [RFC9200]), the requesting entity (the resource server and/or the client) and the authorization server communicate via the token endpoint or introspection endpoint. The use of CoAP and DTLS for this communication is **RECOMMENDED** in this profile. Other protocols fulfilling the security requirements defined in Section 5 of [RFC9200] **MAY** be used instead.

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

If other means of securing the communication with the authorization server are used, the communication security requirements from Section 6.2 of [RFC9200] remain applicable.

7. Security Considerations

This document specifies a profile for the Authentication and Authorization for Constrained Environments (ACE) framework [RFC9200]. As it follows this framework's general approach, the general security considerations from Section 6 of [RFC9200] also apply to this profile.

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

When using pre-shared keys provisioned by the authorization server, the security level depends on the randomness of PSKs and the security of the TLS cipher suite and key exchange algorithm. As this specification targets constrained environments, message payloads exchanged between the client and the resource server are expected to be small and rare. CoAP [RFC7252] mandates the implementation of cipher suites with abbreviated, 8-byte tags for message integrity protection. For consistency, this profile requires implementation of the same cipher suites. For

application scenarios where the cost of full-width authentication tags is low compared to the overall amount of data being transmitted, the use of cipher suites with 16-byte integrity protection tags is preferred.

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

Constrained devices that use DTLS [RFC6347] [RFC9147] are inherently vulnerable to Denial of Service (DoS) attacks, as the handshake protocol requires creation of internal state within the device. This is specifically of concern where an adversary is able to intercept the initial cookie exchange and interject forged messages with a valid cookie to continue with the handshake. A similar issue exists with the unprotected authorization information endpoint when the resource server needs to keep valid access tokens for a long time. Adversaries could fill up the constrained resource server's internal storage for a very long time with intercepted or otherwise retrieved valid access tokens. To mitigate against this, the resource server should set a time boundary until an access token that has not been used until then will be deleted.

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

7.1. Reuse of Existing Sessions

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

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

7.2. Multiple Access Tokens

Implementers **SHOULD** avoid using multiple access tokens for a client (see also Section 5.10.1 of [RFC9200]).

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

7.3. Out-of-Band Configuration

To communicate securely, the authorization server, the client, and the resource server require certain information that must be exchanged outside the protocol flow described in this document. The authorization server must have obtained authorization information concerning the client and the resource server that is approved by the resource owner, as well as corresponding keying material. The resource server must have received authorization information approved by the resource owner concerning its authorization managers and the respective keying material. The client must have obtained authorization information concerning the authorization server approved by its owner, as well as the corresponding keying material. Also, the client's owner must have approved of the client's communication with the resource server. The client and the authorization server must have obtained a common understanding about how this resource server is identified to ensure that the client obtains access tokens and keying material for the correct resource server. If the client is provided with a raw public key for the resource server, it must be ascertained to which resource server (which identifier and authorization information) the key is associated. All authorization information and keying material must be kept up to date.

8. Privacy Considerations

This privacy considerations from Section 7 of [RFC9200] apply also to this profile.

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

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

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

9. IANA Considerations

The following registration has been made in the "ACE Profiles" registry, following the procedure specified in [RFC9200].

Name: coap_dtls
Description: Profile for delegating client Authentication and Authorization for Constrained Environments by establishing a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes.
CBOR Value: 1
Reference: RFC 9202

10. References

10.1. Normative References

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

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

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

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

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

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

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

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

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

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

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

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

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9200] 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)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/info/rfc9200>.

[RFC9201] Seitz, L., "Additional OAuth Parameters for Authentication and Authorization for Constrained Environments (ACE)", RFC 9201, DOI 10.17487/RFC9201, August 2022, <https://www.rfc-editor.org/info/rfc9201>.

10.2. Informative References

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

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

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

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

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

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

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

Acknowledgments

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

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

Authors' Addresses

Stefanie Gerdes
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63904
Email: bergmann@tzi.org

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Göran Selander
Ericsson AB
Email: goran.selander@ericsson.com

Ludwig Seitz
Combitech
Djäknegatan 31
SE-211 35 Malmö
Sweden
Email: ludwig.seitz@combitech.com