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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

Table of Contents

1. Introduction

This specification defines a profile of the ACE framework
[I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource
server use CoAP [RFC7252] over DTLS [RFC6347] to communicate. The
client obtains an access token, bound to a key (the proof-of-
possession key), from an authorization server to prove its
authorization to access protected resources hosted by the resource
server. Also, the client and the resource server are provided by the
authorization server with the necessary keying material to establish
a DTLS session. The communication between client and authorization
server may also be secured with DTLS. This specification supports
DTLS with Raw Public Keys (RPK) [RFC7250] and with Pre-Shared Keys
(PSK) [RFC4279].

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

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

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts
described in I-D.ietf-ace-oauth-authz [2].

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

2. Protocol Overview

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

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

```
C                              RS                      AS
| [-- Resource Request --->] |                         |
|                            |                         |
| [<----- AS Information --] |                         |
|                            |                         |
| --- Token Request  ----------------------------> |
|                            |                         |
| <-------------------------- Access Token ----- |
|                              + Access Information   |
```
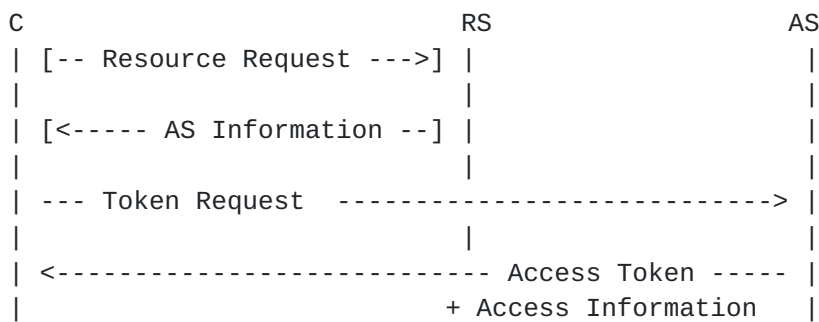
                   Figure 1: Retrieving an Access Token

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

   Once the client knows the authorization server's address, it can send
   an access token request to the token endpoint at the AS as specified
   in I-D.ietf-ace-oauth-authz [6]. As the access token request as well
   as the response may contain confidential data, the communication
   between the client and the authorization server MUST be
   confidentiality-protected and ensure authenticity. C may have been
   registered at the AS via the OAuth 2.0 client registration mechanism
   as outlined in Section 5.3 of draft-ietf-ace-oauth-authz [7].

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

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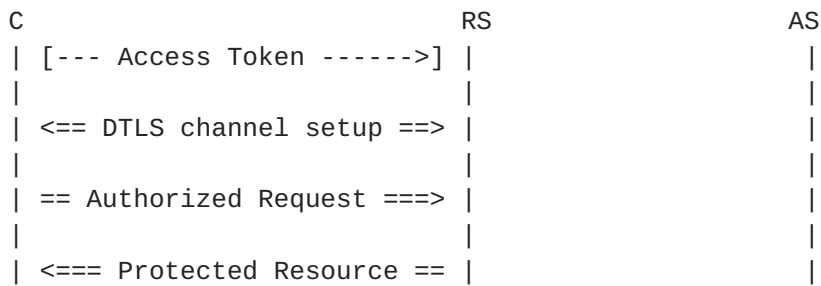   the client C has retrieved the access token from the authorization
   server AS.

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

Figure 2: Protocol overview

3. Protocol Flow

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

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

3.1. Communication between C and AS

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

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

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

3.2. RawPublicKey Mode

After C and AS mutually authenticated each other and validated each
other's authorization, C sends a token request to AS's token
endpoint. The client MUST add a "cnf" object carrying either its raw
public key or a unique identifier for a public key that it has
previously made known to the authorization server. To prove that the
client is in possession of this key, C MUST use the same keying
material that it uses to secure the communication with AS, e.g., the
DTLS session.

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

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

Figure 3: Access Token Request Example for RPK Mode

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

AS MUST check if the client that it communicates with is associated
with the RPK in the cnf object before issuing an access token to it.
If AS determines that the request is to be authorized according to
the respective authorization rules, it generates an access token
response for C. The response SHOULD contain a "profile" parameter
with the value "coap_dtls" to indicate that this profile must be used
for communication between the client C and the resource server. The

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

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

3.2.1. DTLS Channel Setup Between C and RS

Before the client initiates the DTLS handshake with the resource
server, C MUST send a "POST" request containing the new access token
to the authz-info resource hosted by the resource server. If this
operation yields a positive response, the client SHOULD proceed to
establish a new DTLS channel with the resource server. To use the
RawPublicKey mode, the client MUST specify the public key that AS
defined in the "cnf" field of the access token response in the
SubjectPublicKeyInfo structure in the DTLS handshake as specified in
RFC 7250 [10].

An implementation that supports the RPK mode of this profile MUST at
least support the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
[RFC7251] with the ed25519 curve (cf. [RFC8032], [RFC8422]).

Note: According to RFC 7252 [11], CoAP implementations MUST support
   the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251] and
   the NIST P-256 curve. As discussed in RFC 7748 [12], new ECC
   curves have been defined recently that are considered superior to
   the so-called NIST curves. The curve that is mandatory to
   implement in this specification is said to be efficient and less
   dangerous regarding implementation errors than the secp256r1 curve
   mandated in RFC 7252 [13].

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

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

3.3. PreSharedKey Mode

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

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

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

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

                Figure 4: Example Access Token Response

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

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

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

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

   o PartyUInfo = PartyVInfo = (null, null, null)

   o keyDataLength is a uint equal the length of the key shared between
      AS and RS in bits

   o protected MUST be a zero length bstr

   o other is a zero length bstr

   o SuppPrivInfo is omitted

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

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

        Figure 5: Key Derivation Specification in an Access Token

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

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

          Figure 6: Example Access Token Response With Reject

3.3.1. DTLS Channel Setup Between C and RS

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

   C checks if the payload of the access token response contains an
   "access_token" parameter and a "cnf" parameter. With this
   information the client can initiate the establishment of a new DTLS
   channel with a resource server. To use DTLS with pre-shared keys,
   the client follows the PSK key exchange algorithm specified in
   Section 2 of RFC 4279 [20] using the key conveyed in the "cnf"
   parameter of the AS response as PSK when constructing the premaster
   secret.

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

Note: As stated in Section 4.2 of RFC 7925 [21], the PSK identity
   should be treated as binary data in the Internet of Things space
   and not assumed to have a human-readable form of any sort.

If a resource server receives a ClientKeyExchange message that
contains a "psk_identity" with a length greater zero, it uses the
contents as index for its key store (i.e., treat the contents as key
identifier). The resource server MUST check if it has one or more
access tokens that are associated with the specified key.

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

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

Note2: According to RFC 7252 [22], CoAP implementations MUST support
   the ciphersuite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]. A client is
   therefore expected to offer at least this ciphersuite to the
   resource server.

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

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

3.4. Resource Access

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

With the successful establishment of the DTLS channel, C and RS have
proven that they can use their respective keying material. An access
token that is bound to the client's keying material is associated
with the channel. Any request that the resource server receives on
this channel MUST be checked against these authorization rules. RS
MUST check for every request if the access token is still valid.
Incoming CoAP requests that are not authorized with respect to any
access token that is associated with the client MUST be rejected by
the resource server with 4.01 response as described in Section 5.1.1
of draft-ietf-ace-oauth-authz [24].

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

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

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

3. The request is destined for the resource server.

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

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

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

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

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

The client cannot always know a priori if an Authorized Resource
Request will succeed. If the client repeatedly gets error responses
containing AS Information (cf. Section 5.1.2 of draft-ietf-ace-
oauth-authz [26]) as response to its requests, it SHOULD request a
new access token from the authorization server in order to continue
communication with the resource server.

4. Dynamic Update of Authorization Information

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

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

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

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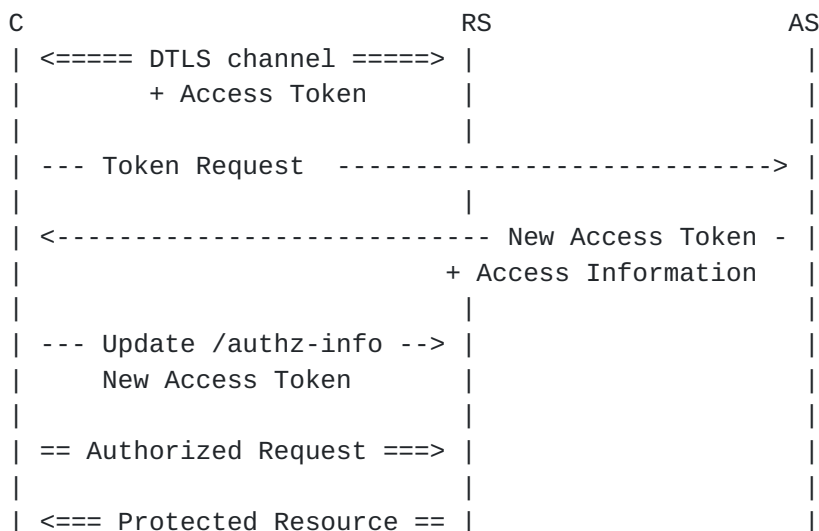   existing session can be used with updated permissions.

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

              Figure 7: Overview of Dynamic Update Operation

5. Token Expiration

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

   As specified in Section 5.8.3 of draft-ietf-ace-oauth-authz [28], the
   resource server MUST notify the client with an error response with
   code 4.01 (Unauthorized) for any long running request before
   terminating the session.

   Table 1 updates Figure 2 in Section 5.1.2 of draft-ietf-ace-oauth-
   authz [29] with the new "kid" parameter in accordance with [RFC8152].

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

                 Table 1: Updated AS Information parameters

6. Security Considerations

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

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

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

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

7. Privacy Considerations

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

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

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

8. IANA Considerations

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

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

Profile name: coap_dtls

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

Profile ID: 1

Change Controller: IESG

Reference: [RFC-XXXX]

9. References

9.1. Normative References

[I-D.ietf-ace-oauth-authz]
          Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
          H. Tschofenig, "Authentication and Authorization for
          Constrained Environments (ACE) using the OAuth 2.0
          Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-16
          (work in progress), October 2018.

[I-D.tiloca-tls-dos-handshake]
          Tiloca, M., Seitz, L., Hoeve, M., and O. Bergmann,
          "Extension for protecting (D)TLS handshakes against Denial
          of Service", draft-tiloca-tls-dos-handshake-02 (work in
          progress), March 2018.

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

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

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

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

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

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

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

9.2. Informative References

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

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

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

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

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

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

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

9.3. URIs

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Authors' Addresses

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

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

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

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

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

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

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se