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

J. Cuellar
P. Kasinathan
Siemens AG
D. Calvo
Atos Research and Innovation
January 2, 2018

Privacy-Enhanced-Tokens (PAT) profile for ACE
draft-cuellar-ace-pat-priv-enhanced-authz-tokens-06

Abstract

This specification defines PAT, "Privacy-Enhanced-Authorization-Tokens", an efficient protocol and an unlinkable-token construction procedure for client authorization in a constrained environment. This memo also specifies a profile for ACE framework for Authentication and Authorization. The PAT draft uses symmetric cryptography, proof-of-possession (PoP) for a key owned by the client that is bound to an OAuth 2.0 access-token.

Status of This Memo

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

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

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

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

Copyright Notice

Table of Contents

1. Introduction

Three well-known problems in constrained environments are the authorization of clients to access resources on servers, the realization of secure communication between nodes, and the preservation of privacy. The reader is referred for instance to [I-D.ietf-ace-actors], [I-D.ietf-ace-oauth-authz] and [KoMa2014]. This memo describes a way of constructing Tokens from an initial secret that can be used by clients and resource servers (or in some cases, more generally by arbitrary nodes) to delegate client authentication and authorization in a constrained environment to trusted and unconstrained authorization servers.

This draft uses the architecture of [draft-ietf-ace-actors] and [I-D.ietf-ace-oauth-authz], designed to help constrained nodes in authorization-related tasks via less-constrained nodes. Terminology for constrained nodes is described in [RFC7228]. A device (Client) that wants to access a protected resource on a constrained node (Resource Server) first has to gain permission in the form of a token from the Authorization Server. This memo also specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz].

The main goal of the PAT is to present methods for constructing authorization tokens efficiently with privacy features such as unlinkability. The CoAP protocol [RFC7252] MAY be used as the application layer protocol. The draft uses symmetric Proof-of-Possession keys [I-D.ietf-oauth-pop-architecture], CBOR web tokens (CWT) [draft-ietf-ace-cbor-web-token-05] claims to represent security claims together with CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] and Concise Binary Object Representation (CBOR) [RFC 7049].

2. Terminology

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

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

- Access-Token (AT): the access token is a token prepared by the AS for C. It represents the privileges granted by the RO to the C to perform actions over the Resources (R) on an RS.

- Token (Tk): this token is prepared by the C, presented to the RS to access the resources (R) on RS. The Tk contains all information needed by the RS to verify that it was granted by AS. The Client derives Tk from the AT.

In version-5 of PAT draft the token names -- AT and Tk -- and their purposes were harmonized with [I-D.ietf-ace-oauth-authz].

3. PAT Overview and Features

The PAT protocol is designed to work with ACE framework [I-D.ietf-ace-oauth-authz] and ACE actors [I-D.ietf-ace-actors]. In this specification we assume the following:

- o A Resource Server (RS) has one or more resources (R) and it is registered with an Authorization Server (AS)
- o The Authorization Server (AS) provides access-tokens for the clients to access resources of RS. The corresponding Resource Owner (RO) of the RS MAY assign allowed-permissions for the Clients in the AS.
- o The RS is offline after commissioning, i.e., RS cannot make any introspective queries to the AS to verify the authorization information provided by the C.
- o A Client (C) is either registered with an AS or it knows how to reach the RS for accessing the required resources.
    - * To access a resource on a Resource Server (RS), a Client (C) should request an access-token (AT) from AS, either directly or using its Client Authorization Server (CAS). For the sake of simplicity, this memo does not include the actor CAS.

Based on the above assumptions, a simple PAT message flow can be described as follows: a C may perform a resource-request to RS without a valid access-token, the RS will reject and it may provide AS information to the C in the response. The C performs an Access-Request to AS to ask for an AT that allows accessing the required resource (R) on RS. The AS checks if C is allowed to access the resource (R) on RS or not, based on permissions assigned by the RO. If C has sufficient permissions, then AS generates an Access-Token (AT) plus proof-of-possession (PoP) key bounded to the access-token and a common secret (K) between AS and RS. AS sends both the AT and the PoP key to C via a secure channel. How this secure channel is created between AS and C is out of the scope of this draft. After receiving AT and PoP key, C performs a resource-request to RS by constructing token (Tk) from AT or by deriving Token. The RS can construct its own version of the PoP key from the AT and verifies the received AT. If it is valid, RS encrypts the response with the PoP key. At the end of this phase, both C and RS has established a common derived secret, the PoP key. Later, C can generate unlinkable tokens (Tk) from the initial AT as described in Section 4.9.

In particular, PAT is designed to be used in contexts where unlinkability (privacy) and efficiency are the main goals: the tokens

(Tk) convey only the assurance of the authorization claims of the clients. In particular, the procedure described in Section 4.9 enables the Tokens (Tk) to be constructed in such a way that they do not leak information about the correspondence of messages to the same Client or from the same access-token (AT). For example, if an eavesdropper observes the messages from different Clients to and from the Resource Servers, the protocol does not give him information about which messages correspond to the same Client. Of course, other information like the IP-addresses or the contents themselves of the requests/responses from lower-layer protocols may leak some information, and this can be treated separately via other methods.

The main features of PAT protocol are described below:

- The PAT method allows a RO, or an Authorization Server (AS) on its behalf, to authorize one or several clients (C) to access resources (R) on a constrained Resource Server (RS). The C can also be constrained devices. The Access-Token (AT) response from AS to C MUST be performed via secure channels.

- The RO is able to decide (if he wishes: in a fine-grained way) which client under which circumstances may access the resources exposed by the RS. This can be used to provide consent (in terms of privacy) from RO.

- The Access-Tokens (AT) are crafted in such a way that the client can derive Tokens (Tk) that allow demonstrating to RS its authorization claims. The message exchange between C and RS for the presentation of the tokens MAY be performed via insecure channels to enforce efficiency. But the payload content -- if the Client is performing a POST/PUT/DELETE request -- from C to RS or the response payload from RS to C MUST be encrypted.

- The RS can derive the PoP key from the AT, which is received attached to the Resource Request message, and it encrypts the response using it.

- The tokens (Tk) do not provide any information about any associated identities such as identifiers of the clients, of access-tokens (AT) and of the resource-server.

- The tokens (Tk) are supported by a "proof-of-possession" (PoP) key and the initial access-token (AT). The PoP key allows an authorized entity (a client) to prove to the verifier (here, the RS), that C is indeed the intended authorized owner of the token and not simply the bearer of the token.

To be coherent with ACE Authorization framework [I-D.ietf-ace-oauth-authz], this draft also specifies an ACE profile to use PAT and for efficient encoding it uses CWT and COSE. The PAT profile is signaled when the C requests token from the AS or via RS in response to unauthorized request response. The PAT profile will cover all the requirements described in [I-D.ietf-ace-oauth-authz].

4. PAT Protocol

The detailed description of PAT protocol is presented in this Section 4. The PAT protocol includes three actors: the RS, the C, and the AS. PAT message flow is shown in Figure 1. Messages in [square brackets] mean they are optional.

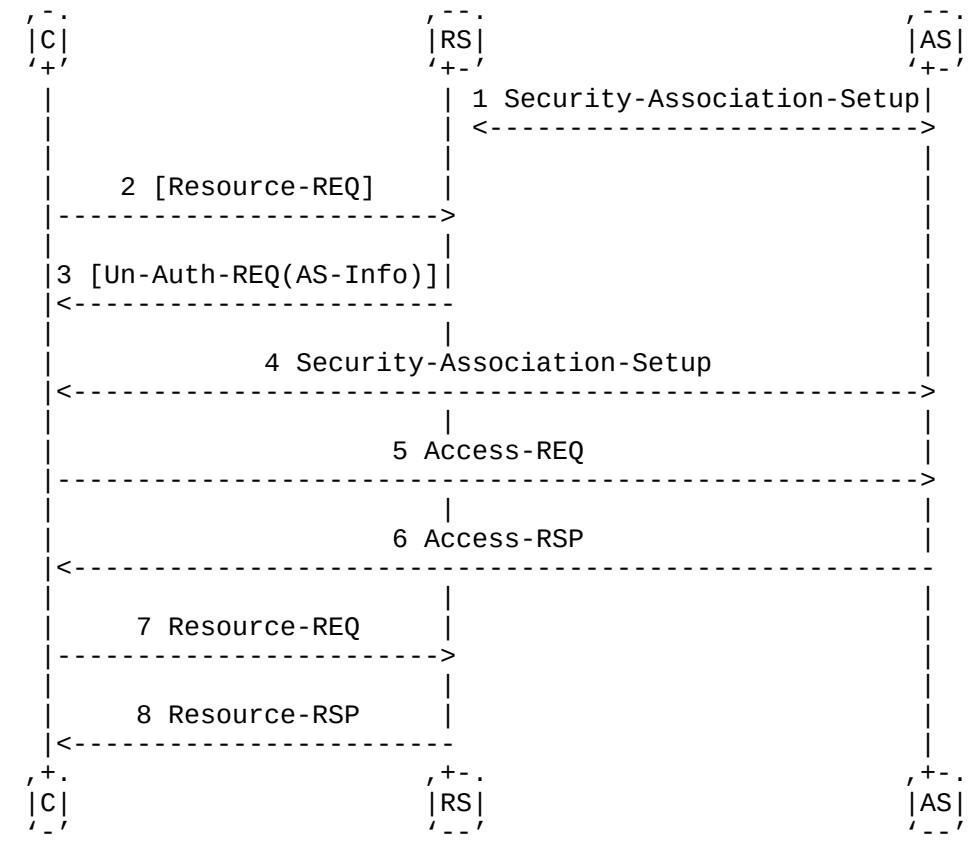


Figure 1: PAT protocol message flow

The following subsections describe the message flow in more detail, especially how the messages and tokens with PoP are constructed.

A PAT message sent from actor A to actor B is represented using the following notation: "A -> B : Message Name"

4.1. RS<->AS: Security-association-Setup

This memo assumes that the Resource Server (RS) and its Authentication Server (AS) share a long term shared secret (K), i.e., a shared key which MAY be implemented via USB (out of band methods) when device commissioning -- out of scope --. The shared secret (K) is used both by the AS and the RS to create proof-of-possession keys (PoP keys or verifiers).

We can also assume that the CAS and AS share a secure connection if CAS exist as an actor, e.g., DTLS. During the commissioning phase, RS registers the cryptographic algorithms and the parameters it supports. The internal clock of RS can be synchronized with the AS during the commissioning phase. Also, PAT supports the use of Lightweight Authenticated Time (LATe) Synchronization Protocol [I.D-draft-navas-ace-secure-time-synchronization].

4.2. [C->RS : Resource-Request]

Initially, a C may not have a valid access-token (AT) to access a protected resource (R) hosted in RS. The C might not also know the corresponding AS-information to request AT from AS. In this scenario, C may send a Resource-Request message to RS without a valid Token (Tk).

To enable resource discovery, RS may expose the URI "/.well-known/core" as described in [RFC6690], but this resource itself MAY be protected. Thus, C can optionally make a CoAP GET request to the URI "/.well-known/core".

4.3. [RS->C : Un-Authorized-Request(AS-Info)]

Once RS receives a resource request from a C, it checks:

- If C has attached a valid token (Tk) or not to the request. If C does not have a valid token (Tk), then RS MUST respond to C with 4.01 (Unauthorized request). Optionally, RS may include information about AS (AS-Info) which includes additional parameters (AS address) to reach the /token endpoint exposed by the AS. Note: this message is sent to any unauthorized Client, therefore it is recommended to include as less information as possible to identify AS.

- If C has a valid access token, but not for the requested resource then RS MUST respond with 4.03 (Forbidden)

- If C has a valid access token, but not for the method requested then RS MUST respond with 4.05 (Method Not Allowed)

- If C has a valid access token, then RS must follow the procedure described in Section 4.8 to create a valid response to C.

Figure 2 shows the sequence of messages with detailed CoAP types between C and RS for the above Un-Authorized-Request:

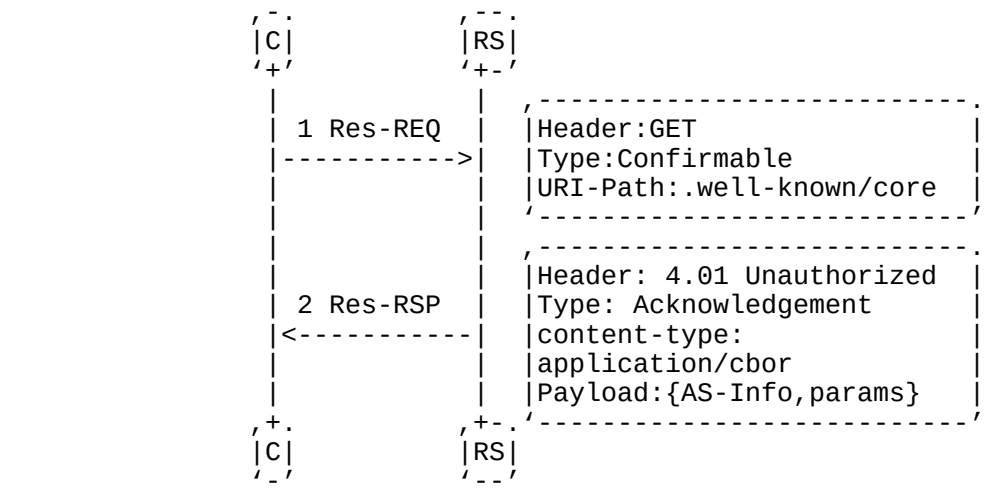


Figure 2: C<->RS Resource-Request and Unauthorized as response

The RS MAY send an Unauthorized response with additional information such as AS-Info and parameters (params). To mitigate attacks based on time synchronization, the Lightweight Authenticated Time (LATe) synchronization protocol [I.D-draft-navas-ace-secure-time-synchronization] MAY be used. In section 6.2 of [I.D-draft-navas-ace-secure-time-synchronization] Possible Scenarios, the scenario 1.2 of suits PAT protocol, an example of it is shown in figure 3.

The response payload MAY include AS information (AS-info) and LATe time synchronization's TIC information object such as key-reference ID (kid) shared secret between RS and AS, a nonce to prevent replay attacks and the message authentication codes (MAC) algorithm [optional] used for producing the MAC. It is recommended for RS to create a MAC tag for TIC parameters.

Figure 3 shows RS example response message to C encoded using CBOR [RFC7049] with pat-type="UnAuthReq".

```
Header: 4.01 (Unauthorized)
Content-Type: application/cbor+pat;
              pat-type="UnAuthReq"
Payload:
{
 AS-Info: "coaps://as.example.com/token",
 #protected
 TIC params:
 {
  nonce: 'rs-nonce..',
  kid: '..',
  [alg]: '..'
  TAG: '..'
 }
}
```

Figure 3: AS information + LATe time synchronization payload

4.4. C<->AS : Security-Association-Setup

Before sending an access-request message, C must establish a secure channel with the AS. The C may be registered with the AS, as described in [I-D.ietf.ace-oauth-authz] or the C MAY have received AS-Info from RS as the answer to a previous Un-Authorized-Request.

The AS may have an access-control list defined by the RO for the authorized clients. With this access-control list, AS can verify if the client is allowed to establish a secure connection or not. If RO granted enough privileges to the client to access the requested resource (R) in RS, then AS establishes a confidential channel with C. How this secure connection (e.g., a DTLS channel) should be established is out of the scope of this memo.

Notice that, it is important to ensure that the connection between AS and C MUST be reliable and secure since the PAT protocol relies on the fact that the messages exchanged between C and AS are protected and confidential. If the Client is also a constrained device, then C may use DTLS-profile as described in [I.D-draft-gerdes-ace-dtls-authorize] to create the secure channel with the AS.

4.5. C->AS : Access-Request

Once C establishes a secure communication channel with AS, C sends an access-request (ACC-REQ) message to AS to the endpoint /token requesting an access token for RS as described in [I-D.ietf.ace-oauth-authz].

Optionally, the C includes as part of the Access-Request Message the details about the resources (R) or their corresponding URI with the operations it needs to access or perform on RS. If not AS should prepare an access token with default permissions. Fine grained access to resources (R) of RS depends on the infrastructure or services the RS offers. For example, if RS exposes different resources such as temperature and humidity, a generic access token may be granted by AS to C to access both resources on RS. On the other hand, the application developer or administrator may decide the access-rights based on application requirements.

Figure 4 shows an access-request message sent from C to AS to get an access token. The "aud" represents a specific resource R ("tempSensor") and "scope" represents the allowed actions that C aims to perform as described in [I-D.ietf-ace-oauth-authz] using CWT [I-D.ietf-ace-cbor-web-token]. The Scope parameter can be designed based on application requirements i.e., it can be "read" or "write" or methods such as "GET|POST" etc. If RS has included TIC information for time synchronization, then the C MUST include TIC object, including the MAC -- if included -- without any changes in the payload for access request.

How the client authenticates itself against the AS is out of the scope of this draft. Nevertheless, in the following example, the client presents the Client_Credentials i.e., password based authentication by presenting its client_secret (see section 2.3.1. of [RFC6749]).

```
Header: POST (Code=0.02)
Uri-Host: "coaps://as.example.com"
Uri-Path: "token"
Content-Type: "application/cbor+cwt+late ;
late-type=tic"
Payload:
{
  "grant_type" : "client_credentials",
  "client_id": "client123",
  "client_secret": "Secret123",
  "aud" : "tempSensor",
  "scope": "GET|POST",
  ... omitted for brevity ...
  TIC params:
  {.. [if exist] ..
  nonce:'rs-nonce..', # same rs-nonce sent by RS
  kid: '..'
  }
  TAG: .. # TIC MAC tag produced by RS
            using the shared key k with AS.
}
```

Figure 4: Example Client Access-Request message to AS

4.6. C<-AS : Access-Response

When AS receives an access-request message from a C, AS validates it and performs the following actions:

- If the access request from C is valid (i.e., operations are covered by the privileges defined by the RO), then AS prepares the Access-Token (AT) and sends it with COAP response code 2.01 (Created).

- If the Access-Request from C contains LATe time synchronization TIC information object, then an appropriate response with TOC information object is included in the response as described in [I.D-draft-navas-ace-secure-time-synchronization].

- If the client request is invalid then AS MUST send appropriate COAP error response code as specified in [I-D.ietf-ace-oauth-authz].

The Figure 5 shows the Access-Request from C to AS and the Access-Response from AS to C.

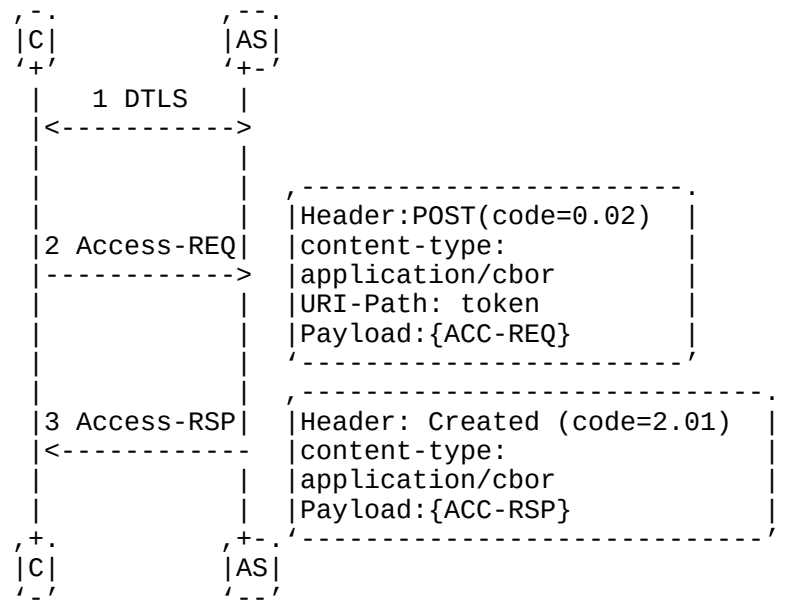


Figure 5: Access-Request and Access-Response

The AS constructs the Access-Token (AT) and the verifier (the symmetric PoP key) as the answer for a valid access request from C. The contents of the access-response (ACC-RSP) payload are logically split into two parts: the Access-Token (AT) and the Verifier (Symmetric PoP key).

4.6.1. Access-Token construction:

- The Access-Token is constructed by AS using the CWT claim parameters. It represents the permissions granted to the Client.
  - "iss" (issuer): AS identity
  - "aud" (audience): resource server URI or specific resource URI for a fine-grained procedure.
  - "exp" (Expiration Time): token expiration time
  - "iat" (Issued At): token issued at time by AS
  - "cti" CWT ID should be unique for every Access-Token.
  - "scp" (Scope): Note that scp is not a CWT claim. It can specify allowed methods such as GET, POST, PUT or DELETE.

Other CWT claims are optional. It is recommended to avoid the CWT claim "sub" (subject) as it exposes client identity.

4.6.2. Verifier or PoP key construction:

- Verifier (Symmetric PoP key): G (K, Access-Token). The Client will use the Verifier as the key material to communicate with the RS, i.e., if C wants to encrypt its payload, it uses the verifier as the key.
  - G: the MAC algorithm which is used to create the verifier, we propose Poly1305 [RFC7539]. Notice that G is a function which takes two parameters (key, data) as input and it produces a keyed digest as the output
  - K: the shared key between AS and RS
  - Access-Token: constructed using CWT claims as explained before

It is of special importance that the Access-Response message with the access token and the verifier MUST be sent to C through a secure channel -- in our example we considered a DTLS channel between C and AS --.

The time-synchronization between AS and RS MAY be implemented based on the application requirements using [I.D-draft-navas-ace-secure-time-synchronization].

The AS should specify required parameters as described in [I-D.ietf-ace-oauth-authz] such as the type of token, etc. Also, if the Access-Request from C does not include any profile, AS MUST signal the C to use appropriate or default profile that is supported by RS.

If the access-request message includes LATe TIC information, then AS MUST prepare TOC information and included it in the response. A MAC tag for TOC is created and appended in the response to prevent the client from tampering TOC information.

Figure 6 shows the example of an Access-Response sent from AS to C after successful validation of C’s credentials which were presented using an Access-Request message.

```
Header: 2.01 (Created)
Content-Type: application/cbor+cwt+pat; pat-type="tk"
Location-Path: token/...
Payload:
{
   "access token": b64'SlAV32hkKG ...
   {
    "iss": https://as.example.com
    "aud": "tempSensor",
    "scp": "read",
    "iat": 1...,
    "cti": "..", # Unique can be a Sequence Number
    "exp": 5...,
    "alg": "chacha20/poly1305",
    "profile": "ace_pat"
   }
  "cnf":
  {
   COSE_Key: {
     "kty": "symmetric",
     "kid": h'...
     "k": b64'jb3yjn...  #[verifier]
   }
  }
 TOC:{
   as_time: '..',
   nonce: 'rs-nonce..',
   }
 tag: '..' #TOC tag
}
```

Figure 6: Example Access-Response message sent from AS to C with detailed CWT params and payload info

4.7. C->RS : Resource-Request

Once C receives the Access-Response from AS, C can construct a token (Tk) which will demonstrate that C has got the sufficient authorization to access resources (R) in RS.

A new Token (Tk) MUST be attached to each RES-REQ sent to RS by C. If payload data are included, then C should encrypt them using the verifier as key and optionally it can include an Authentication Hash (AuthHash= Hash(verifier+C_nonce)). PAT profile provides necessary recommendations by using AEAD (e.g., chach20/poly1305) algorithm.

o As an example if C performs:

* A CoAP GET() without payload. In this case, the request from C MAY be sent un-encrypted since it does not include confidential data, but the response from RS with payload MUST be always encrypted and only the valid C MUST be able to decrypt it.

* A CoAP POST(), a CoAP PUT() or a CoAP DELETE() request with payload MUST be protected and encrypted by using the AEAD algorithm specified in the Access Token (AT). PAT profile proposes to use ChaCha20-Poly1305-AEAD authenticated encryption mechanism, while using the Verifier (PoP key) as the key and a nonce. The AuthHash MAY be protected by using it as Additional Authentication Data (AAD) in the AEAD algorithm.

The RS MUST implement /authz-info endpoint to allow any Client to transfer the token (Tk) as described in [I-D.ietf-ace-oauth-authz]. The Resource-Request message with valid Token (Tk) shall be constructed from AT by C and it should be sent to RS in the following way:

o Figure 7 shows the example of Client Resource-Request:

```
Request:
Content-Type: application/cose+cbor+pat;
           pat-type="AuthReq";
Message:
{  CoAP request: GET/POST/PUT/DELETE
   Uri-Host "coap://rs.example.com"
   uri-path: /authz-info
   payload:
   {
     Token:
        {
         Access Token(AT), # Tk encapsulates the AT from AS
         AuthHash=Hash(verifier+nonce), #optional for GET
         nonce,
         #Chach20/Poly1305(Verifier,nonce,
                AAD=AuthHash, payload)
         Payload:{
             # if exist
             }
        }
    }
}
```

Figure 7: RES-REQ from C using /authz-info implemented at RS

Figure 7 shows the detailed example of GET RES-REQ to the endpoint /authz-info implemented at RS as described in [I-D.ietf-ace-oauth-

authz]. This option enables the C to transport the token (Tk) to the RS. After receiving the request, RS verifies the token (Tk): RS can construct its own version of verifier or PoP-key by performing G(K,AT) from the access-token (AT); and RS checks whether AuthHash=Hash(verifier+nonce) is valid or not. If Tk and AuthHash are valid, then RS sends an encrypted response using the verifier (PoP key).

o Figure 8 shows the GET request from C to RS described in [I-D.ietf-ace-oauth-authz], with pat-type="AuthReq".

```
Header: GET
Content-Type: application/cose+cbor+pat;
              pat-type="AuthReq";
Uri-Host: "coap://rs.example.com"
Uri-Path: /authz-info
Payload:
{ token: {
   "access token": .. {
     "aud": "tempSensor"
     "scp": "read"
     ... #CWT omitted for brevity.
   }
   "nonce": ..
   "AuthHash": .. #[AuthHash=hash(verifier+nonce)]
  }
  TOC:{
  time:'as-time',
  nonce:'rs-nonce',# rs-nonce from RS TOC object
  } tag: '..' #TOC tag
}
```

Figure 8: Example of valid GET RES-REQ from C to RS including time-sync using endpoint /authz-info.

The C performs a GET request to "tempSensor" using CWT claim "aud", and together C also transfers the Token (Tk) to the RS. PAT allows performing both RES-REQ and transferring authorization information in RES-REQ. In the next example we show how to perform a resource request if the C performs a POST request with encrypted payload information.

o Figure 9 shows an example of POST Resource-Request from C to RS described in [I-D.ietf-ace-oauth-authz], with pat-type="AuthReq".

```
 Header: POST (Code=0.02)
 Content-Type: application/cose+cbor+pat;
               cose-type="encrypt0";
               "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /authz-info
Payload:
{# COSE
    token: {
    "access token": .. {
        "aud": "firmwareUpd"
        "scp": "write"
        ... CWT omitted for brevity,
        }
    "nonce": ..  # nonce
    "AuthHash": .. #[AuthHash=hash(verifier+nonce)]
     TOC:{
         time:'as-time',
         nonce:'rs-nonce', # rs-nonce from RS TIC
         } tag: '..' #TOC tag
    }
# COSE_Encrypt0 + COSE_MAC0 Protected
    ciphertext:{
    #Chacha20/Poly1305 AEAD payload using
        # key=verifier,
        # nonce=..,
        # AAD=AuthHash
        },
    tag: ..
}
```

Figure 9: Example of valid POST request from C to RS

Figure 9 shows the POST Resource-Request from C to RS where the Uri-Path "/authz-info" allows the authorized client to perform firmware upgrade on the RS using the CWT claim "aud:firmwareUpd". PAT recommends protecting sensitive information such as the payload using AEAD algorithm (chacha20/poly1305). The client should use Verifier or PoP key as the key, a nonce, and AuthHash as AAD.

4.8. RS->C : Resource-Response

When the RES-REQ with a token (Tk) arrives from C to RS, RS MUST evaluate the resource request and the token (Tk) in the following order:

- Step 0: Check whether the contents of Tk are derived from an access-token (AT) or not.

- Step1: If Tk contains the access-token (AT) from AS, extract AT. Extract nonce and Authentication Hash (AuthHash) from the request message.
  - Step1.1: (If available) Verify the freshness of the sequence number (cti) in the access token presented by AS.
  - Step1.2: Generate the verifier by computing G(K, access token) where K is the shared key between AS and RS.
  - Step1.3: Compute a verification hash as Hash(verifier+nonce) and compare the result with AuthHash for correctness.
  - Step1.4: Check if the access token has valid CWT parameters such as "aud", "scp", "exp", "nbf", etc for the requested resource or action to be performed.
  - Step1.5: (IF available) Synchronize RS internal clock using TOC object as described in [I.D-draft-navas-ace-secure-time-synchronization].
- Step2: If the token is valid, RS should create a temporary internal state as shown in table 1 below with details of CWT claims "cti","exp","scp"", and the verifier (PoP key).

The RS internal state table which is shown in Table1 also includes "next cti". The next cti (cti x) value is computed as the Hash of previous cti (cti x-1) and the verifier. The purpose of this is explained in the section Section 4.9.

Verifier	cti_x-1	exp	scp	next cti (cti_x)
G(k,AT)	cti_x0= cti of AT	of AT	of AT	cti_x1= hash(cti_x0,Verifier)

Table 1: RS Internal state table of access-tokens and RS_nonce

- Step 3: If the token is valid, then RS decrypts the payload from the client (if exist) Verifier (PoP key).
- Step 4: After that, RS prepares the response and encrypts the payload with a fresh nonce, PoP key. Only the Client (C) with a valid key (the Verifier) can decrypt the payload:

4.8.1. RS Response-codes to C RES-REQ:

- If the token (Tk) is valid -- as discussed above --, then RS MUST respond with payload-data as described above with the appropriate response code as described in [RFC7252]. For example, to a POST request with 2.01 (created) or 2.04 (changed).

- If the token (Tk) is invalid, then RS MUST respond with code 4.01 (Unauthorized)

- If the token (Tk) is valid but does not match the "aud" or resource C is requesting for then RS MUST respond with code 4.03 (Forbidden)

4.9. Construction of Derived-Tokens (DT)

The objectives to create Derived-Tokens (DT) are:

- To produce Unlinkable Tokens (Tk). It is not efficient for the client to request a new access-token (AT) from AS everytime. Also, if C uses the same access-token (AT) from AS, the identity of the client can be identified via the AT CWT claim "cti" (token identity).

- To reduce token (Tk) size (efficiency in transport) that the client must send to RS /authz-info in every resource request.

- To create tokens (Tk) that may have limited access to protected-resources -- fine-grained resource access tokens -- from the original access-tokens (AT) that could grant more privileges to protected-resources on RS. For example, an access-token (AT) could provide permissions to access all protected-resources on RS via CWT claims audience "aud" and scope "scp". The client could derive a Token (Tk) providing access to a reduced set of protected-resources available on RS from the initial AT.

4.9.1. C->RS: Resource-Request via DT

The Client receives an encrypted response from RS after its first RES-REQ with the access-token (AT) from AS.

The Client creates a new Derived-Token(DT) using CWT claims as described below. In order to minimize the data size, we use only the claims which are required.

- Client MAY prepare a DT with a subset of scope "scp" operations that the client received from the initial Access-Token (AT). It creates the first derived "cti_x1" by Hash("cti_x0 + verifier")

from the CWT claim "cti" of the original access-token (AT). The subsequent derivation of "cti_x" can be performed by a generic function "cti_x = Hash(cti_x-1 + verifier)". Note that the derived-token (DT) MUST include all the necessary CWT claims such as "cti_x", "aud", "exp", "scp". All other CWT claims are optional.

- Client creates the AuthHash=(verifer+nonce).
- Client prepares encrypted content using verifier as the key -- if there is any payload --.
- Note: in the Additional Authenticated data (AAD), the C includes AuthHash and the derived-token (DT), so that the payload cannot be misused/exchanged with another RES-REQ or nonce.

```
 Header: POST (Code=0.02)
 Content-Type: application/cbor+cwt+cose++pat;
              cose-type="encrypt0";
              "pat-type="AuthReq";
Uri-Host: "coap://rs.example"
Uri-Path: /firmware
Payload:
{# COSE
    token: {derived-token(DT):
        "aud": "firmwareUpd",
        "exp": ..
        "scp": "write",
        "cti": Hash(cti_x+verifier)
        # cti_x=Hash(cti_x-1+verifier).
        }
    "nonce": ..  # new nonce
    "AuthHash": h'bfa03.. #[Hash=(verifier+nonce)]
# COSE_Encrypt0 + COSE_MAC0 Protected
    ciphertext:{
    #Chacha20/Poly1305 AEAD payload using
        # key=verifier,
        # nonce=..,
        # AAD=AuthHash,DT
        h'....omitted for brevity
    },
    tag: h'... omitted for brevity
}
```

Figure 12: Example of valid Resource-Request from C to RS using a derived-token(DT)

4.9.2. RS->C : Resource-Response to DT

After receiving the Token (Tk) which encapsulates the derived Token (DT) from C, RS performs the following Steps. If any of them fails, then RS must send an UnAuthorized response to C, and C must use the first AT, which was received from the AS, or request a new AT based on the resource owner (RO) configuration:

- RS extracts CWT claim cti (cti_x) from the Derived-Token (DT) and checks if it exists in its internal state table. If RS finds the cti_x, then RS uses the corresponding verifier, "cti_x-1, "exp", and "scp" to perform the validation of next steps.
- RS checks that cti_x= Hash (cti_x-1+verifier)
- RS checks that AuthHash == Hash(verifier+nonce)
- RS checks that the permissions are valid using "scp" and expiration time "exp"
- RS updates the new cti_x-1, cti_x in its internal state table
- RS creates an encrypted response to be sent to C with a payload including payload-data.

msg#	Verifier (V)	cti_x-1	exp	scp	cti_x= Hash(cti_x-1+V)
0	G(K,AT)	0x00	of AT	of AT	0xAB = Hash(0x00+V)
1 (upd)	G(k,AT)	0xAB	of AT	of AT	0xFF = Hash(0xAB+V)

Table 2: RS updating only two parameters in its internal stating table 1

The Table 2 shows the RS internal state table with an example.

5. Security Considerations

TBD

5.1. Privacy Considerations

The CoAP messaging layer parameters such as token and message-id can be used for matching a specific request and response. TBD

6. IANA Considerations

TBD

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7252] Shelby, Z., Hartke, K. and Borman, C., "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC6347] Rescorla E. and Modadugu N., "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7539] Y. Nir and A. Langley: ChaCha20 and Poly1305 for IETF Protocols, RFC7539, May 2015

[I-D.ietf-ace-actors] Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-0 (work in progress), March 2017.

[I-D.ietf-oauth-pop-architecture] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authorization for the Internet of Things using OAuth 2.0", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

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

[I.D-draft-navas-ace-secure-time-synchronization] Navas, G., Selander, G., Seitz, L., "Lightweight Authenticated Time (LATe) Synchronization Protocol", draft-navas-ace-secure-time-synchronization-00 (work in progress), October 2016.

7.2. Informative References

[KoMa2014] Kohnstamm, J. and Madhub, D., "Mauritius Declaration on the Internet of Things", 36th International Conference of Data Protection and Privacy Comissioners, October 2014.

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

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

[I.D-draft-gerdes-ace-dtls-authorize] Gerdes, S., Begmann, O., Bormann, C., Selander, G., Seitz, L. Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE), draft-gerdes-ace-dtls-authorize-01, March 2017.

[I-D.ietf-ace-cbor-web-token] Jones, M., Tschofenig, H., Erdtman, S., CBOR Web Token (CWT), draft-ietf-ace-cbor-web-token-05 (work in progress), June 2017..

8. Acknowledgement

This draft is the result of collaborative research in the RERUM EU funded project and has been partly funded by the European Commission (Contract No. 609094). The authors thank Ludwig Seitz for reviewing the previous version of the draft.

8.1. Copyright Statement

Appendix A. ACE profile Registration

TBD

ACE profile template	PAT
Profile name	TBD
Profile Description	TBD
Profile ID	TBD

Table2: ACE profile registration template

Authors' Addresses

Jorge Cuellar
Siemens AG
Otto-Hahn-Ring 6
Munich, Germany 81739

Email: jorge.cuellar@siemens.com

Prabhakaran Kasinathan
Siemens AG
Otto-Hahn-Ring 6
Munich, Germany 81739

Email: prabhakaran.kasinathan@siemens.com

Daniel Calvo
Atos Research and Innovation
Poligono Industrial Candina
Santander, Spain 39011

Email: daniel.calvo@atos.net

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2017

T. Hardjono
MIT
N. Smith
Intel Corp

July 4, 2016

Fluffy: Simplified Key Exchange for Constrained Environments
draft-hardjono-ace-fluffy-03

Abstract

This document proposes a simplified key exchange protocol for the establishment of a symmetric key shared between two devices or entities within a constrained environment. The pair-wise key establishment is performed using the mediation of a trusted Simple Key Distribution Center (SKDC) entity. The protocol also supports the mediated distribution of a group-key among multiple devices or entities for the purposes of protecting multicast messages.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 5, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document proposes a simplified key exchange protocol for constrained environments for the establishment of a symmetric key shared between two constrained devices. The pair-wise key establishment is performed using the mediation of a trusted Simple Key Distribution Center (SKDC) entity. The protocol also supports the mediated distribution of a group-key among multiple devices or entities for the purposes of protecting multicast messages.

The simplified key exchange protocol is referred to here as "Fluffy" and is based on a reduced set of Kerberos [RFC4120] messages, adjusting the message flows, types and features to the needs and capabilities of constrained devices and environments. It does not seek to be backward compatible with Kerberos implementations.

The protocol aims to be independent of the underlying transport protocol, and as such the protocol messages are integrity-protected against modifications in-transit. Similar to Kerberos [RFC4120], messages that carry sensitive information (such as keys and/or keying material) are protected using authenticated-encryption. Non-sensitive fields of the messages are integrity-protected using checksums or keyed-hash in the manner of RFC3961. A separate specification will be developed to address in more detail these cryptographic aspects of the current proposed protocol.

Two families of protocol messages are defined here:

- Pairwise key establishment between two entities: When a client seeks to establish a pairwise shared key (called the session encryption key) with a service principal (SP), it invokes the mediation of the SKDC. A four (4) message flow among the client, SKDC and SP are used to establish the pairwise shared key. A further two messages are used to delete the key prior to its expiration.

- Group-shared key establishment among multiple entities: When a client (e.g. client#1) seeks to create a group-shared key (called the group encryption key), it invokes the SKDC to create the group-key, to retain a copy at the SKDC and to return a copy to the requesting client. The distribution of the group-key to other members of a multicast group uses a simple fetch/deliver model in

which new group members (e.g. client#2) must ask for a copy of the group-key from the SKDC.

An additional set of exchanges are introduced to support the delivery of a public key pair to a client entity, with or without an accompanying digital certificate.

The current simplified key exchange protocol does not address the initial secret establishment between an entity and the SKDC. This is referred to in RFC4120 and RFC6113 as "pre-authentication". We anticipate that many types of constrained devices would need to undergo "on-boarding" into an operational state within a constrained environment, and that the on-boarding process may include (directly or as a side-effect) the establishment of the initial secret between the new device and the SKDC already operating in the environment. Thus, for example, the on-boarding process of a device (e.g. door-lock) into a constrained environment (e.g. home basement) with an SKDC entity (e.g. within the alarm panel) may consist of the device and the SKDC running a Diffie-Hellman exchange with the assistance of the human owner. The topic of on-boarding and off-boarding of devices is outside the scope of the current specification.

In this specification we assume that a transport such as CoAP [RFC7252] will be deployed in constrained environments where the IP protocol is operating at the network layer. Environments that are using non-IP transport are out of scope currently for this specification.

The current protocol uses JSON [RFC7159] and CBOR [RFC7049] for its message format. This is in-line with the RESTful paradigm and provides the greatest flexibility for the current protocol to be integrated with other protocols such as OAuth2.0 [RFC6749] for authorization and UMA [UMACORE] for user-centric consent management.

Since the intended deployment environment for the current protocol is a constrained environment, devices and entities there are assumed to use the UUID as the basis for identification. How a device is assigned a UUID is out of scope for the current specification.

The current specification acknowledges that in certain types of constrained environments there is the need for devices to not only operate autonomously for long periods of time, but also for devices to have the capability to take-on different roles with respect to other devices in the environment. Thus, a device (D1) acting as a client to another device (D2) that is acting as an SKDC could also be acting as an SKDC for yet a third device (D3). Thus, the device D1 may have the capability to be both a client and SKDC depending on the operational environment.

As in many deployment environments generally, often security is a trade-off among several factors (e.g. usability, assurance levels, cost, economic risk/benefits, and others). As such, it is realistic to acknowledge that the degree of trustworthiness of an SKDC is dependent on the value of the data and connections within the deployment environment. Thus, an SKDC within a home environment may not be expected to feature the same level of resistance to attacks as an enterprise deployment of a Kerberos KDC.

1.1. Notational Conventions

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

Unless otherwise noted, all protocol properties and values are case sensitive. JSON [JSON] data structures defined by this specification MAY contain extension properties that are not defined in this specification. Any entity receiving or retrieving a JSON data structure SHOULD ignore extension properties it is unable to understand. Extension names that are unprotected from collisions are outside the scope of this specification.

1.2. Terminology

The current specification seeks to share terminology as much as possible with the terminology defined for CoAP [RFC7252]. However, since the intended Application(s) play a crucial role within constrained networks, we also refer to terminology used by OAuth 2.0 and UMA. Note that within a given constrained network, an device make take multiple roles (client or server) depending on the exchange and layers of the exchange in which they participate.

client
: The client is the entity in a constrained environment seeking to share a key pair-wise with the service principal.

service principal
: The entity with whom the client seeks to establish a pair-wise symmetric key is refereed to as the service principal (SP). This terminology is used to avoid confusion as much as possible with the generic term "server" or "service".

simple key distribution center
: The simple key distribution center (SKDC) is the entity which mediates the establishment of the pair-wise shared key between the client and service principal.

miniticket
: This is the data structure that carries the symmetric key to be shared between the client and service principal.

receipt
: This is the data structure that carries the symmetric key from the SKDC to an entity (client or service principal).

authenticator
: This is the data structure that carries proof-of-possession of a shared symmetric key between two entities.

pair-wise shared key
: A pair-wise shared key (PSK) is symmetric key shared only between a client and service principal.

group shared key
: A group shared key (GSK) is symmetric key shared by two or more entities.

session encryption key
: The session encryption key is the symmetric key generated by the SKDC to be shared pair-wise between the client and the service principal. A session encryption key is an instance of a PSK.

group encryption key
: The group encryption key is the symmetric key generated by the SKDC to be shared among members of a multicast group. A group encryption key is an instance of a GSK.

secret key
: The secret key is the symmetric key that is uniquely shared pair-wise between a client (or service principal) and the SKDC. This term is borrowed from RFC4120. Thus, the client secret key is the symmetric key that is uniquely shared pair-wise between the client and the SKDC. The SP secret key is the symmetric key that is uniquely shared pair-wise between the SP and the SKDC.

set-up keying material
: The cryptographic keying material (including possibly keys) resulting from the initial on-boarding process of a device into a constrained environment is referred to generally as "set-up keying material".

permissions and access control

The permissions and access control (PAC) is the set of information pertaining to permissions for entities within a constrained environment.

resource
The resource refers to the end-point at the service principal to which the application seeks access.

1.3. Design Considerations and Assumptions

There are a number of design considerations and background for the current protocol.

Transport: We assume that the entities in the constrained environment are deploying the CoAP protocol as transport [RFC7252]. However, the design of the current protocol seeks to be transport-independent as much as possible because we anticipate that not all constrained networks may be running CoAP.

JSON data structures: The data structures in this specification are expressed in JSON. We believe this provides the greatest flexibility for the protocol to be integrated into existing protocols for authorization (such as OAuth2.0 [Oauth2] and OpenID-Connect [OIDC]) and consent management by the resource/device owner (such as the User Managed Access (UMA) protocol [UMACORE]).

On-boarding and off-boarding: We assume that constrained devices will undergo the phase of "on-boarding" into a constrained environment. Similarly, "off-boarding" will be required when a constrained device leaves (or is removed) from a constrained environment. The notion of on-boarding is closely related to that of "take-ownership" of certain types of devices. Technologies such as the TPM [TPM] and EPID [EPID] play a crucial role in providing both cryptographic proof (technical trust) and human proof (social trust) in the process of changing the ownership of a device when it is activated and introduced into a constrained environment. We see a close relationship between on-boarding and the current protocol for establishing PSKs and GSKs within a constrained environment.

Secret key establishment or derivation: Following the on-boarding process of a client (resulting in the client and SKDC possessing set-up keying material), the client and the SKDC are assumed to generated the secret key which is shared pair-wise between the client and the SKDC. Methods include using PRFs and other one-way functions. The exact process of generating a secret key from the set-up keying material is out of scope of the current specification. As such, the current Fluffy protocol begins with

the assumption that each entity (client and service principal) already shares pair-wise a secret key with the SDKC. This secret key should be used only for key-management related messages as defined in this specification. Additionally, in this specification we have avoided the use of the term "long-term key" to refer to this secret key due to the broad meaning of this term.

Realms and zones: We have borrowed the notion of "realms" from RFC4120 because we anticipate that a constrained environment may consists of one or more physical networks, and which may be arranged (logically or physically) into "zones". Furthermore, we anticipate that in some use-cases the notion of a "realm" or "zone" may be more ephemeral than is commonly understood for RFC4120 deployments. Thus, there may be constrained use-cases where realms or zones are short-lived.

1.4. Out of Scope and Non-Goals:

The following are out of scope (non-goals) for the current specification:

Authorization and permissions: The issue of permissions and authorization is out of scope for this specification. However, the current specification anticipates the close integration between permissions, authentication and key establishment.

Discovery: Discovery of endpoints, identities, services and other aspects of a constrained environment is out of scope for the current specification.

Backward compatibility with Kerberos: It is not a goal of this specification to achieve backward compatibility with RFC1510 or RFC4120. Similarly, it is not the goal of this specification to be compatible with the MS-PAC [MSPAC] and MS-KILE [MSKILE] specifications.

Pre-authentication: RFC4120, RFC4556 and RFC613 uses the term "pre-authentication" to denote a client obtaining keying material for its secret key prior executing the Kerberos. It is not a goal of this specification to address pre-authentication.

Channel Binding for TLS and DTLS: Channel binding [RFC5929] for DTLS or TLS are out of scope in the current specification.

Certificate Issuance and Management: Issuance of X509 digital certificates and certificate management (in the sense of RFC2459, RFC2797 and RFC4210) is out of scope in the current specification.

2. Common Building Blocks

The current protocol employs a number of data structures that are common across several message types. A number of these data structures have semantic equivalents in RFC4120, while some are newly introduced.

Depending on the message type, some fields may be overloaded in its usage. For example, in the PSK-Request message the client states the identity and realm of the service principal within the SKDC-REQ-BODY. This is similar to RFC4120 because three (3) parties are involved in a PSK establishment (initiated by the client sending a PSK-Request message to the SKDC). However, when the SKDC-REQ-BODY is used in GSK establishment (initiated by an entity sending the SKDC a GSK-Request (GSK-REQ) message) the identity and realm fields are used instead to communicate the desired identity and the realm of the multicast group.

Two building blocks that do not have equivalents in RFC4120 are the Key Data and the Key Envelope structures:

Key Data: The keydata structure is used to convey cryptographic key(s) together with the associated operational parameters for the key(s). The structure of the keydata follows the JSON Web Key (JWK) definition of keys and keying material [RFC7517]. This is a departure from RFC4120.

Key Envelope: The key envelope is used to convey parameters related to a key (e.g. KeyID) but not the key itself. It is used in cases where entities need to refer to a key (e.g. group-key) without having to carry the key in the message.

In the following the common building blocks are discussed.

2.1. SKDC Request Body

The SKDC request body (SKDC-REQ-BODY) carries specific information regarding the type of request and the entities involved in the message. This data structure is used in the initial request send from a client (or SP) to the SKDC.

The SKDC-REQ-BODY varies slightly when used in the PSK-REQ and GSK-REQ messages.

SKDC-REQ-BODY:

- Key Type (kty): This denotes the type of key being requested by the sender (client or SP) of the request.

- Desired algorithm (etype): This is the algorithm that is desired (or supported) by the sender of the request (client or SP) .

- SKDC options - optional (skdc-options): These are flags that are intended for the SKDC only. This field is optional.

- SKDCs realm (skdcrealm): This the name of the realm, domain or zone of the SKDC.

- SKDC's identity (skdcname): This is the identity of the SKDC.

- Service principal's realm - optional (sprealm): This the name of the realm, domain or zone of the service principal in a PSK-REQ message. In a GSK-REQ message it is the realm of the multicast group. This field is optional.

- Service principal's identity (spname): In a PSK-REQ message this is the identity of the service principal. In a GSK-REQ message it is the identity or name of the multicast group.

- Client permissions - optional (cpac): In a PSK-REQ message this is the permisions desired by the client for itself. In a GSK-REQ message this is the permisions desired by the client for the multicast group. This field is optional.

2.2. Miniticket

The miniticket is always created by the SKDC and is always intended for the service principal (although it is delivered via the client who initiates with the PSK-REQ message). The SKDC responds to the client by sending a PSK-Response (PSK-REP) message containing the miniticket and a receipt. The miniticket contains a copy of the session encryption key to be delivered to the service principal by the client in a PSK-Establish (PSK-ESTB) message. As such, the sensitive parts (enc-part) of the miniticket is encrypted using the service principal's secret key (which it shares pair-wise with the KDC). The miniticket is functionally equivalent to the service-ticket in RFC4120.

The miniticket contains the following:

- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this miniticket.

- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this miniticket.

- Service principal's realm - optional (sprealm): This the name of the realm, domain or zone of the service principal for whom this miniticket is destined. This field is optional.

- Service Principal's identity (spname): This is the identity of the service principal for whom this miniticket is destined.

- Encrypted miniticket part (enc-part): This is the encrypted part of the miniticket intended for the service principal. It is encrypted using the secret key shared pair-wise between the SKDC and the service principal. The encrypted part contains the following:

  * Ticket flags - optional (tflags): This is the flags set by the SKDC for the service principal concerning this ticket. This field is optional.

  * Client's realm (crealm): This the name of the realm, domain or zone of the client with whom the receiver of this miniticket shares the enclosed key.

  * Client's identity (cname): This is the identity of the client with whom the receiver of this miniticket shares the enclosed key.

  * Client permissions - optional (cpac): This is the permissions and access control (PAC) structure containing permissions granted to the client associated with the enclosed key. This field is optional.

  * Time of authentication (authtime): This is the time at the SKDC when it created this miniticket in response to a request.

  * Expiration time of key (endtime): The is the expiration time of the key in this miniticket.

  * Service principal permissions - optional (sppac): This is the permissions and access control (PAC) structure granted to the service principal associated with the enclosed key. This field is optional.

  * Transited realms - optional (transited): This is the set of SKDCs and realms that was involved in the issuance of the miniticket for the service principal. This field is used for cross-realm ticket issuance. This field is optional.

  * Key data (keydata): This fields contains the key-data structure that carries the cryptographic key and other parameters

necessary for operating the key. See section Section 2.6 for the key-data structure.

2.3. Receipt

The receipt is always created by the SKDC and is used for the SKDC to deliver a key to a requesting party (be it client, service principal or multicast group members). The receipt is functionally equivalent to the SKDC-Response part in RFC4120.

In The PSK estabslishment flows the receipt is used to deliver the new session encryption key to the requesting client in a PSK-REP message from the SKDC.

In The GSK establishment flows the receipt is used to deliver the new group encryption key to the requesting entity using the GSK-Response (GSK-REP) message. Similarly, members of a multicast group must individually request a copy of the group key using the GSK-Fetch message (GSK-FET), to which the SDKC will send a receipt structure containing a copy of the group key via the GSK-Deliver (GSK-DLVR) message.

When used in a PSK-REP message as a response to the client’s request for a new session encryption key, the receipt names the service principal who is to share the key with the client. The service principal identified in this receipt is the same as that stated in the matching miniticket.

When used in a GSK-REP message for a new group-key creation, the receipt instead names the multicast group with associated with this key. Note that the GSK-REP message has no accompanying miniticket because the SKDC is reponding solely to the requester of the new group-key in a 2-party flow. The receipt in a GSK-Deliver (GSK-DLVR) message (to deliver a copy of the group-key to members of the multicast group) is functionally identical to the receipt in the GSK-REP message.

A note about convention: in the remainder of this specification the entity that requests the creation of a group-key is denoted as the service principal (SP). The members of the multicast group are denoted as the client.

Receipt:

- Receipts flags - optional (rflags): This is the flags set by the SKDC concerning this receipt. This field is optional.

- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this receipt.

- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this receipt.

- Realm - optional (sprealm or mcastrealm): This field is optional, and is used as follows:

  * sprealm: In a PSK-REP message, sprealm is used. This is the name of the realm, domain or zone of the service principal with whom the client (recipient of this receipt) shares the enclosed session encryption key.

  * mcastrealm: In a GSK-REP message and GSK-DLVR message, the mcastrealm is used. This is the realm of the multicast group associated with the enclose group encryption key.

- Name of entity sharing this key (spname or mcastname):

  * spname: In a PSK-REP message, spname is used. This is the identity of the service principal who will be sharing the enclosed session encryption key with requesting client.

  * mcastname: In a GSK-REP message and GSK-DLVR message, mcastname is used. This is the identity of the multicast group associated with the enclosed group encryption key.

- Service Principal's SKDC - optional (spskdc): This field is optional. In a PSK-REP message, this is the identity of the service principal's SKDC. This field is absent in receipts used in a GSK-REP message or GSK-DLVR message.

- Permissions - optional (cpac or grppac): This field is optional. This is the permissions and access control (PAC) structure containing permissions granted, associated with the enclosed key.

  * cpac: In a PSK-REP message, the permissions pertains to the client who requested the session encryption key.

  * grppac: In a GSK-REP message or GSK-DLVR message, the permissions pertain to the members of the multicast group who share this common group encryption key.

- Time of authentication (authtime): This is the time at the SKDC when it created this receipt.

- Nonce from the sender’s request (nonce): This is the nonce found in the previous request message (either PSK-REQ message or GSK-REQ message).

- Expiration time of key (endtime): The is the expiration time of the key in this receipt.

- Key data (keydata): This fields contains the key-data structure that carries the cryptographic key and other parameters necessary for operating the key. In a PSK-REP message, this key is the session encryption key to be shared between the client and service principal. In a GSK-REP message or GSK-DLVR message this is the group encryption key to be shared by the multicast group members. See section Section 2.6 for the key-data structure.

2.4. Authenticator

The authenticator is used by a sender to provide proof-of-possession (POP) to a receiver of a given key that the sender shares with the receiver. The authenticator here is functionally equivalent to the authenticator in RFC4120.

In the PSK-REQ and GSK-REQ messages, the requesting entity uses the authenticator to "authenticate" itself to the SKDC by providing proof-of-possession of the secret key which it shares pair-wise with the SKDC. Note that this mode of usage of the authenticator departs from RFC4120 where a key request message (namely the AP-REQ in RFC4120) is not accompanied by an authenticator.

In the PSK establishment flows, the authenticator is used also in the PSK-ESTB message that is sent from the client to the service principal. More specifically, in the PSK-ESTB message the client encrypts the authenticator using the session encryption key that the client obtained in the previous PSK-REP message it received from the SKDC. Here the authenticator proves to the service principal that the client knows the session encryption key that is enclosed in the accompanying miniticket.

The authenticator is also used in the PSK deletion and GSK deletion flows where the SKDC accompanies its PSK-DELT and GSK-DELT messages respectively with an authenticator that proves to the intended recipient of the message that the SKDC knows the secret key shared pair-wise with that recipient. As such, the authenticator can be used as a generic mechanism to provide proof-of-possession of a shared key between two entities.

Using the example of a client sending an authenticator to a service principal, the authenticator contains the following:

- Client’s realm (crealm): This the identity of the realm, domain or zone of the client that created the authenticator.
- Client’s identity (cname): This is the identity of the client that created the authenticator.
- Client’s current time (ctime): This is the time at the client when it created the authenticator.
- Nonce (nonce): This is a new nonce generated by the client (for the recipient of the authenticator).
- Sequence number - optional (seqnum): This is the sequence number used by the client to detect attacks. This field is optional.
- Checksum - optional (cksum): This is the keyed-checksum (based on the session encryption key) used by the client as sender. This field is optional.

2.5. Acknowledgement

The acknowledgement structure (ack) is used by one entity to send a positive acknowledgement to another entity regarding a message that was previously exchanged. The acknowledgement would carry a nonce that was found in the related previous message. The type of acknowledgement is expressed in the enveloping header message (e.g. PSK-ACK type acknowleges a previous PSK-ESTB message).

Note that the acknowledgement structure is intended to be generic, and as such can also be used by the SKDC to send an acknowledgement to a client or service principal.

Using the example of a service principal (sender) sending the an acknowledgement to a client (receiver), the acknowledgement structure contains the following:

- Service principal’s realm - optional (sprealm): This the identity of the realm, domain or zone of the service principal who created this acknowledment. This field is optional.
- Service Principal’s identity (spname): This is the identity of the service principal for who created this acknowledgement.
- Nonce from the client’s previous message (nonce): This is the nonce found in the previous message being acknowledged.
- Time of authentication (authtime): This is the time at the service principal when it created this acknowledgement message.

- Sequence number - optional (seqnum): This is the sequence number used to detect attacks. This field is optional.

2.6. Key Data

The key-data structure is used to carry cryptographic keys and the related parameters needed to operate the keys. The construction of the key-data structure follows that of the JSON Web Keys [RFC7517] which supports not only symmetric keys, but also public key pairs and X509 certificates.

The intent is to support the delivery of either a single symmetric key, a public key pair or one key (half) of a public key pair.

Delivery of an array of keys is for future consideration.

2.6.1. Symmetric Key Data

The key-data structure for carrying a symmetric key is as follows.

- Key type (kty): This is the type of key contained in the current key-data structure.
- Key ID - optional (keyid): This is the name or handle for the key that can be be referred to by parties sharing they key. This field is optional.
- Key (key): This is the symmetric key.
- Key operations parameters (keyops): This is parameters required to operate the cryptographic key.
- Algorithm (alg): This is the algorithm name for the use of the key.

2.6.2. Asymmetric Key Data

The key-data structure for carrying an asymmetric key and parameters is as follows.

- Key type (kty): This is the type of key contained in the current key-data structure.
- Key ID - optional (keyid): This is the name or handle for the key that can be be referred to by parties sharing they key. This field is optional.
- Key (key): This is the public key.

- Key operations parameters (keyops): This is parameters required to operate the cryptographic key.
- Algorithm (alg): This is the algorithm name for the use of the key.
- Public key usage (pkuse): For public keys this field indicates the use of the key. See RFC7517.
- X509 URL parameter (x5u): This parameter is a URI [RFC3986] that refers to a resource for an X.509 public key certificate or certificate chain [RFC5280]. See RFC7517.
- X509 certificate chains parameter - optional (x5c): This parameter contains a chain of one or more PKIX certificates [RFC5280]. See RFC7517.
- X509 thumbprint parameter - optional (x5t): This parameter contains a base64url-encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. See RFC7517.
- X509 SHA256 thumbprint parameter - optional (x5t256): This parameter contains a base64url-encoded SHA-256 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. See RFC7517.

2.7. Key Envelope

The key envelope structure is used in messages that refer to a cryptographic key by its KeyID. As such, the key envelope structure by definition must never carry any cryptographic keys or keying material.

The key envelope is used primarily in the deletion of a a PSK or a GSK. When the SKDC wishes to delete a given PSK that it shares with an entity, it can refer to the target key by way of the KeyID in the key envelope.

In order to delete a PSK that a client and service principal shares (through the mediation of the SKDC via a previous 3-party PSK establishment flow), the SKDC must separately delete the PSK at the client and at the service principal (i.e. using two separate PSK-Delete (PSK-DELT) messages).

The key envelope is encrypted using the secret key shared between the SKDC and the entity (client or service principal) for whom the envelope is destined.

The key envelope must never be encrypted using the target key that is to be deleted.

The key envelope structure contains the following:

- Envelope Flags - optional (envflags): This is the flags related to the envelope. This field is optional.
- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this envelope.
- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this envelope.
- Current time (authtime): This is the time at the SKDC when it created the encrypted envelope.
- Nonce (nonce): This is a new nonce generated by the SKDC (for the recipient of the envelope).
- Sequence number - optional (seqnum): This is the sequence number to detect attacks. This field is optional.
- Key data (keydata): This is the key-data structure which contains the KeyID of the target key. The key-data in a key envelope must not contain any cryptographic keys. See section Section 2.6 for the key-data structure.

3. Pair-wise Shared Key Establishment

This section describes the pair-wise key establishment between the client and the service principal.

3.1. Basic Protocol Exchange

Prior to executing the Fluffy protocol, a client must first be in possession of a secret key that it shares pair-wise with the SKDC. The process or method of obtaining the client secret key is outside the scope of the current specification, but for a device operating within a constrained environment this may be a direct consequence of the on-boarding or take-ownership process.

The PSK establishment consists of a 4-message flow from the client to SKDC, the SKDC back to the client, and between the client and the service principal.

Note that unlike RFC4120, the client must provide an authenticator in its first message (PSK-Request) to the SKDC. This authenticator is

encrypted by the client using the secret key it shares pair-wise with the SKDC.

The message flows consists of the following steps and are summarized in Figure 1.

- PSK-Request (PSK-REQ): The client sends a PSK-Request message to the SKDC asking for the SKDC to mediate the sharing of a new session encryption key between the client and the service principal. The client must indicate the intended service principal in this message. Unlike RFC4120 the client must include an authenticator that is encrypted to the SKDC as a proof of possession of the secret key it shares with the SKDC.

- PSK-Response (PSK-REP): The SKDC responds by generating a new session encryption key and placing the key into a miniticket intended for the service principal. The miniticket is encrypted using the secret key which is pair-wise shared only between the SKDC and the service principal. Additionally, the SKDC places a copy of this new session encryption key into a receipt structure, and encrypting it using the secret key pair-wise shared between the SKDC and the client. Both the miniticket and the receipt are then returned to the client. The client must forward the miniticket unmodified to the service principal in the PSK-Establish message.

- PSK-Establish (PSK-ESTB): The client then decrypts the receipt to obtain the session encryption key. The client uses this session encryption key to encrypt an authenticator structure as proof of possession for the service principal. The client then sends the authenticator and the miniticket (unmodified from the SKDC) to the service principal. The service principal decrypts the miniticket to obtain the session encryption key, and then it uses the session encryption key to verify the authenticator (by decrypting it). At this point the client and the service principal shares the session encryption key.

- PSK-Acknowledge (PSK-ACK): The service principal exercises the newly received session encryption key by encrypting an acknowledgement message to the client.

Similar to RC4120, the integrity of the messages containing cleartext data is protected using a checksum mechanism (e.g. keyed hash) based on the client’s secret key [RFC3961].

The PSK Establishment Flows

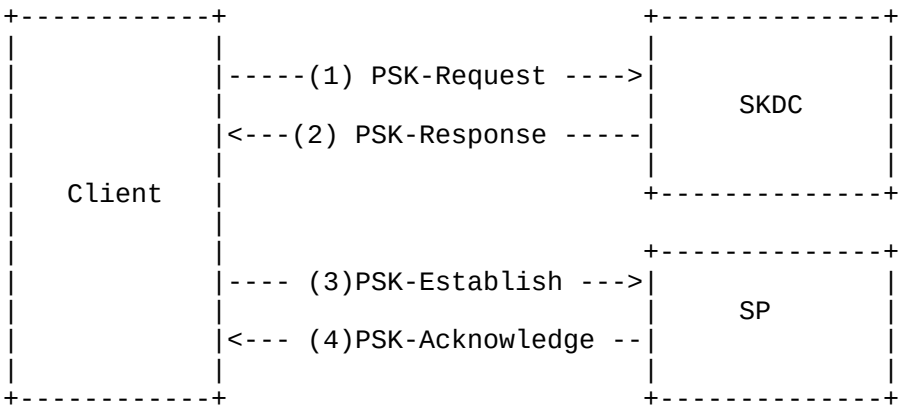


Figure 1

The message components as used in the protocol are summarized in Figure 2. Note that all protocol messages are integrity-protected, and some are encrypted.

The PSK Message Components

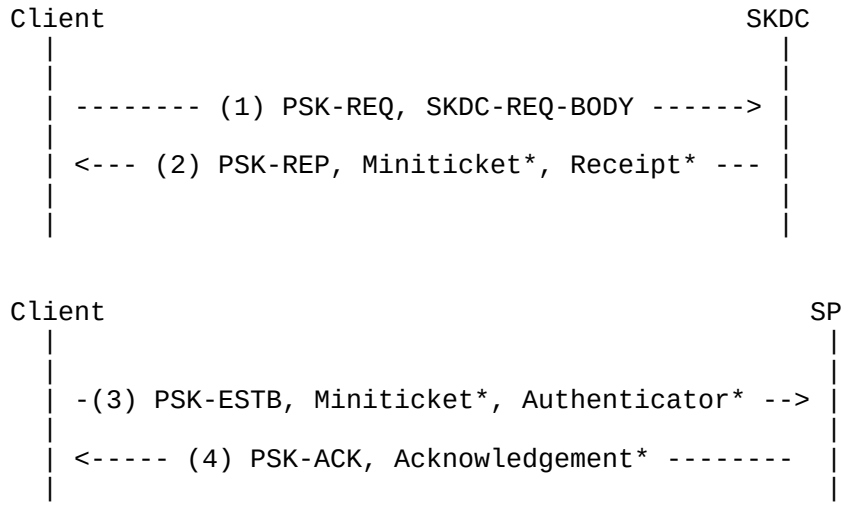


Figure 2

3.2. PSK-Request Message (PSK-REQ)

The PSK-Request (PSK-REQ) message is sent from the client to the SKDC asking for the SKDC to mediate the establishment of a pair-wise shared key between the client and the service principal. The client must indicate the intended service principal in this message.

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-REQ".
- SKDC request body (req-body): The request body contains the parameters required by the SKDC to mediate key establishment for the client. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a PSK-REQ message contains the following:
  - Key Type (kty): This is type of key being requested by the client from the SKDC.
  - Desired algorithm (etype): This is the algorithm that is desired (or supported) by the client.
  - SKDC options - optional (skdc-options).
  - SKDC’s realm (skdcrealm).
  - SKDC’s identity (skdcname).
  - Service principal’s realm - optional (sprealm).
  - Service principal’s identity (spname).
  - Client permissions - optional (cpac).
- Authenticator (authenticator): This is the authenticator encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information on the authenticator structure.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.3. PSK-Response Message (PSK-REP)

The PSK-Response (PSK-REP) is sent by the SKDC to the client as a response to the client's previous PSK-REQ message. The PSK-REP message carries two crucial data structures, namely the miniticket and the receipt.

The miniticket here is intended solely for the service principal and carries a copy of the session encryption key (key) intended for the service principal. As such the miniticket is encrypted to the service principal using the secret key shared between the SKDC and service principal. Although the miniticket is returned to the client, the client is unable to view or modify the encrypted parts of the miniticket.

The receipt here is intended solely for the client and carries a copy of the session encryption key (key) intended for the client. The receipt is encrypted to the client using the secret key shared between the SKDC and client.

The PSK-REP message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-REP".
- Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.
- Client's identity (cname): This is the identity of the client.
- Miniticket (mticket): The miniticket contains the following:
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Service principal's realm - optional (sprealm).
  - Service Principal's identity (spname).
  - Encrypted miniticket part (enc-part): This is the part of the PSK-REP message that is encrypted by the SKDC to the service principal, using the secret key that the SKDC shares pair-wise with the service principal. It contains the following:
    - Ticket flags - optional (tflags).

+ Client’s realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.

+ Client’s identity (cname).

+ Client permissions - optional (cpac).

+ Time of authentication (authtime).

+ Expiration time of this key (endtime).

+ Service principal permissions - optional (sppac).

+ Transited realms - optional (transited).

+ Key data (keydata): This key-data structure contains a copy of the session encryption key destined for the service principal. See section Section 2.6 for the key-data structure.

o Receipt (receipt): This is the part of the PSK-REP message that is encrypted by the SKDC to the client, using the secret key that the SKDC shares pair-wise with the client. It contains the following:

* Receipts flags - optional (tflags).

* Issuing SKDC’s realm (skdcrealm).

* Issuing SKDC’s identity (skdcname).

* Service principal’s realm - optional (sprealm).

* Service Principal’s identity (spname).

* Service Principal’s SKDC - optional (spskdc).

* Client permissions - optional (cpac).

* Time of authentication (authtime).

* Nonce from the Client’s previous PSK-REQ request message (nonce).

* Expiration time of this key (endtime).

    * Key data (keydata): This key-data structure contains a copy of the session encryption key destined for the client. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.4. PSK-Establish Message (PSK-ESTB)

The PSK-Establish (PSK-ESTB) message is sent from the client to the service principal requesting it to share a key (i.e. the session encryption key). The PSK-ESTB message contains two parts. The first is the miniticket obtained by the client from the SKDC in the previous PSK-Response (PSK-REP) message.

The second is the authenticator created by the client. The authenticator is encrypted using the session encryption key which the client obtained in the receipt from the previous PSK-REP from the SKDC).

The authenticator proves to the service principal that the client is in possession of the session encryption key.

This PSK-ESTB message is sent by the client to the service principal. It contains the following:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "PSK-ESTB".

- Client’s realm (crealm): This the name of the realm, domain or zone of the client.

- Client’s identity (cname): This is the identity of the client.

- Miniticket (mticket): This is the miniticket structure (unmodified) that the client received from the SKDC in the previous PSK-REP message. See Section Section 3.3 for the miniticket in the PSK-REP message.

- Authenticator: The authenticator in the PSK-ESTB contains the following:

    * Client’s realm (crealm).

    * Client’s identity (cname).

    * Client’s current time (ctime).

    * A new nonce generated by the client for the service principal (nonce).

    * Sequence number - optional (seqnum).

    * Checksum - optional (cksum).

  o Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.5. PSK-Acknowledge Message (PSK-ACK)

The PSK-Acknowledge (PSK-ACK) message is sent from the service principal to the client in response to the previous PSK-ESTB message.

The message contains an acknowledgement part that is encrypted by the service principal using the session encryption key (which the service principal obtained in the miniticket in the previous PSK-ESTB message).

The PSK-ACK message contains the following:

  o Protocol version (pvno): This the version of the protocol.

  o Message type (msg-type): The message type for this message is "PSK-ACK".

  o Client’s realm (crealm).

  o Client’s identity (cname).

  o Acknowledgement (ack): This is the acknowledgement structure that contains the following:

    * Service principal’s identity (spname).

    * Service principal’s realm - optional (sprealm).

    * Nonce from the Client’s previous PSK-ESTB message (nonce).

    * Time of authentication (authtime).

    * Sequence number (inremented) from PSK-ESTB message - optional (seqnum).

- o Extensions - optional (ext): Reserved for future extensions. This field is optional.

4. Pair-wise Shared Key Deletion

The current protocol supports the proactive deletion by the SKDC of a pairwise shared key (PSK) prior to the expiration of the key. The target PSK to be deleted must be a PSK that a client and service principal had established through the mediation of the SKDC using the PSK establishment flow.

Only the SKDC has the authority to send a key-deletion message (PSK-DELT) to an entity (client or service principal). A client or service principal MUST ignore key-deletion messages which did not come from the SKDC.

The SKDC authenticates itself to the client (service principal) by including a key envelope that is encrypted using the secret key shared between the SKDC and the client (service principal). The key envelope identifies the target key to be deleted by its key-ID.

If a PSK-DELT message issued by the SKDC is received at the client after the named key has in fact expired, the client must still respond with a PSK-Delete-Confirm (PSK-DELC) message. This confirms to the SKDC that the named key no longer exists at the client.

In order to delete a PSK that a client and service principal shares (through the mediation of the SKDC via a PSK establishment flow), the SKDC must delete the PSK at the client and at the service principal separately (using two separate PSK-DELT messages respectively).

The message components as used in the protocol are summarized in Figure 3.

The PSK Delete Message Components

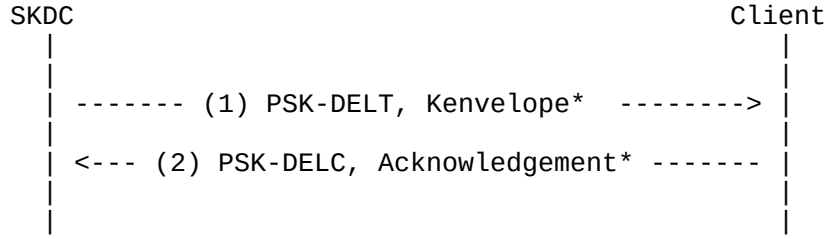


Figure 3

4.1. PSK-Delete Message (PSK-DELT)

The PSK-DELT message is sent from the SKDC to a client (or service principal) asking for the delition or erasure of the PSK identified in the message. The SKDC must indicate the intended recipient in this message.

- Protocol version (pvno): This is the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-DELT".
- Client’s realm (crealm): This the identity of the realm, domain or group in which the client belongs in connection to this request.
- Client’s identity (cname): This is the identity of the client.
- Key Envelope (kenvelope): The key envelope is encrypted using the client’s secret key that it shared with the SKDC. The key envelope contains the following:
  - Envelope Flags - optional (envflags).
  - Issuing SKDC’s realm (skdcrealm).
  - Issuing SKDC’s identity (skdcname).
  - Current time (authtime).
  - New nonce generated by the SKDC (nonce).
  - Sequence number - optional (seqnum).
  - Key data (keydata): This key-data structure contains the KeyID of the target key to be deleted. The key-data in a key envelope must never contain a cryptographic key. See section Section 2.6 for the key-data structure.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

4.2. PSK-Delete-Confirm Message (PSK-DELC)

The psk-delete-confirm (PSK-DELC) message is sent from the client (or service) to the SKDC confirming the removal of the PSK identified in the previous PSK-DELT message. A client (or service principal) must only send the PSK-DELC message after it has successfully remove or erase the target key from its key store.

The acknowledgement in the PSK-DELC message is encrypted by the client using the secret key that the client shares with the SKDC. See Section Section 2.5 for more information on the acknowledgement structure.

The PSK-DELC message contains the following:

- Protocol version (pvno): This is the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-DELC".
- SKDC's realm - optional (skdcrealm).
- SKDC's identity (skdcname).
- Acknowledgement (ack):
  - Client's identity (cname).
  - Client's realm - optional (crealm).
  - Nonce from the SKDC's previous PSK-DELT message (nonce).
  - Client's current time (authtime).
  - Sequence number - optional (seqnum): This field is optional.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5. Group Shared Key Establishment

The current protocol supports the establishment of a group-shared symmetric key (referred to as the "group encryption key" or "group-key") among a number of entities within a constrained environment. The group encryption key affords group-authenticity for messages but not source-authenticity since the symmetric key is shared among multiple entities (members of the multicast group). See RFC3740 for further discussion regarding multicast group security.

In the following we use the notation of the service principal (SP) as the entity that initiates the multicast group creation by requesting the SKDC to create a new group encryption key and to maintain a copy of that group-key until such time it expires or is deleted. The clients that request and obtain a copy of the group-key are denoted as "members" of the group. The current specification follows the

convention that only the group-creator and the SKDC are permitted to proactively delete a group encryption key.

Using the service principal as the group-creator, the service principal must accompany its request to the SKDC with an authenticator.

Each group encryption key is associated with an owner (creator) who requested its creation at the SKDC. When an SP seeks to establish a new group encryption key, it sends a GSK-Request message to the SKDC asking that the SKDC generate a new symmetric key (i.e. the group encryption key), return a copy of the group encryption key to the service principal (via a receipt inside a GSK-Response message) and for the SKDC to retain a copy of the group key (for subsequent fetches by the clients). The sensitive parameters of the GSK-Response message (including the group encryption key) inside the receipt is encrypted using the secret key pair-wise shared between the SP and the SKDC.

When a client seeks to obtain a copy of a group encryption key associated with a multicast group, the client sends a GSK-Fetch message to the SKDC identifying the multicast group (mcastname) of interest. The requesting client must accompany the request with an authenticator that is encrypted using the secret key shared between the client and the SKDC.

If a corresponding group or group encryption key does not exist at the SKDC, the SKDC returns an error message. Otherwise, the SKDC returns a copy of the group encryption key (inside a receipt) to the requesting client using the GSK-Deliver message. The sensitive parameters of the GSK-Deliver message (including the group encryption key) inside the receipt is encrypted using the secret key pair-wise shared between the requesting client and the SKDC.

The GSK Establishment Flows

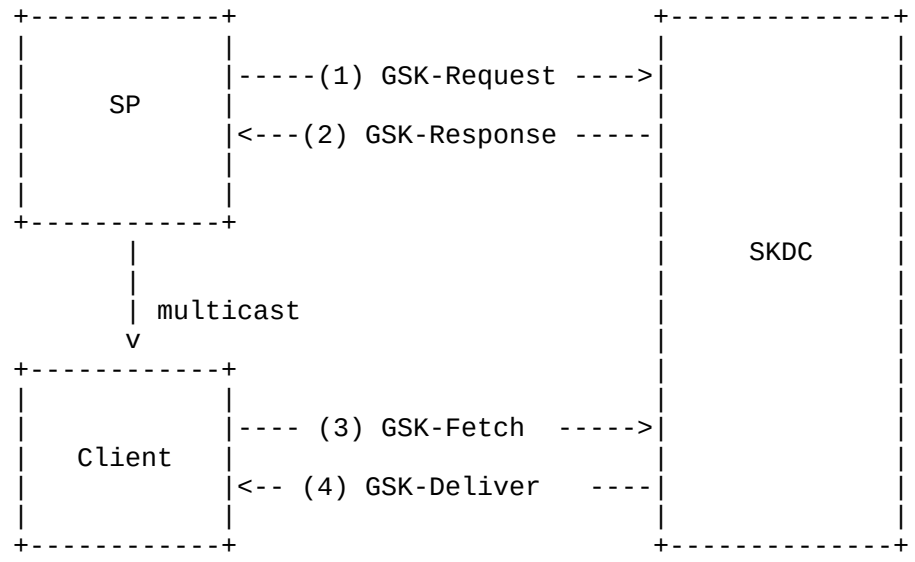


Figure 4

The GSK establishment in the protocol consists of two sets of 2-messages each:

- Creation of the group-key at the SKDC:
  - GSK-Request: A service principal sends a GSK-Request (GSK-REQ) message to the SKDC asking for a new group key to be created. The service principal provides the desired name (mcastname) of the multicast group. It must accompany the request with an authenticator to the SKDC. After generating the new group-key the SKDC retains a copy of the group-key (until it expires) and associates it with the multicast group name (mcastname).
  - GSK-Response: The requesting service principal obtains a copy of the new group-key via the GSK-Response (GSK-REP) message from the SKDC. A receipt structure to carries the group-key. The receipt is encrypted by the SKDC using the secret key it shares with the service principal.
- Fetching of a copy of the group-key from the SKDC:
  - GSK-Fetch: To request a copy of the group-key, a client sends the GSK-Fetch (GSK-FET) message to the SKDC with an

authenticator. The client must indicate the desired multicast group (mcastname) in the GSK-FET message.

* GSK-Deliver: The SKDC returns a copy of the group-key to the client via the GSK-Deliver (GSK-DLVR) message. A receipt structure to carries the group-key. The receipt is encrypted by the SKDC using the secret key it shares with the client.

The message components as used in the protocol are summarized in Figure 5. Note that all protocol messages are integrity-protected, and some are encrypted.

The GSK Message Components

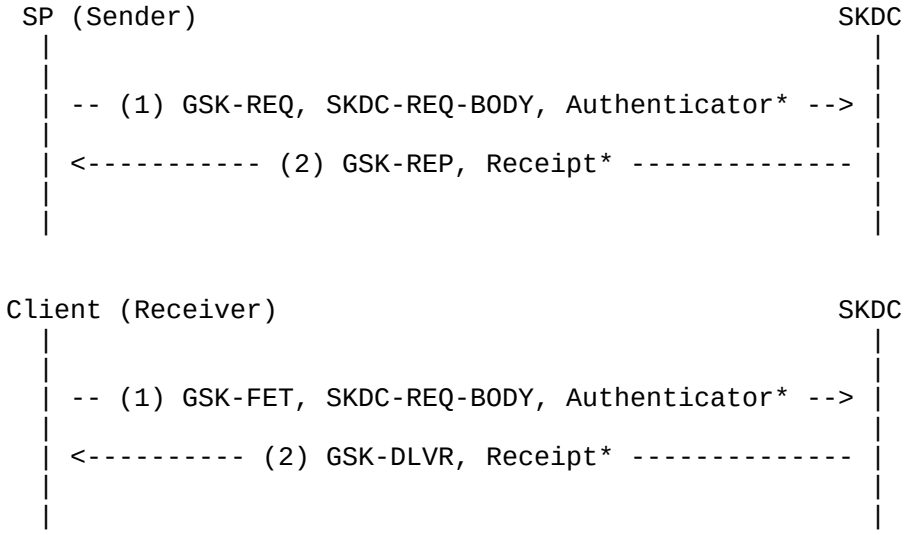


Figure 5

5.1. GSK-Request Message (GSK-REQ)

The GSK-Request message is sent from the service principal to the SKDC asking the SKDC to create a new group-key. The service principal authenticates itself to the SKDC by including an authenticator in the GSK-REQ message.

The contents of the GSK-REQ message is as follows:

o Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-REQ".

- SKDC request body (req-body): The request body contains the parameters related to the group-key and multicast group. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body in a GSK-REQ message contains the following:

  * Key Type (kty): This is type of key being requested. For a group shared key the type is "SYMM".

  * Desired algorithm (etype): This is the algorithm that is desired (or supported) by the service principal.

  * SKDC options - optional (skdc-options).

  * SKDC’s realm (skdcrealm).

  * SKDC’s identity (skdcname).

  * Multicast group realm - optional (mcastrealm): This is the desired realm name associated with the multicast group.

  * Multicast group identity (mcastname): This is the desired identity or name for the multicast group.

  * Group permissions - optional (grppac): This is the desired set of permissions associated with the multicast group.

- Authenticator: In the case of a GSK-REQ message, the authenticator is encrypted by the service principal (SP) using the secret key it shares with the SKDC. The authenticator in the GSK-REQ contains the following:

  * SP’s realm (sprealm).

  * SP’s identity (spname).

  * SP’s current time (sptime).

  * A new nonce generated by the SP (nonce).

  * Sequence number - optional (seqnum).

  * Checksum - optional (cksum).

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.2. GSK-Response Message (GSK-REP)

The GSK-Response (GSK-REP) message is sent from the SKDC to the service principal in response to the service principal's GSK-Request message. The GSK-Response message contains a receipt structure which carries the new group-key for the requesting service principal.

Note that the GSK-REP message does not contain a miniticket.

The Receipt in the GSK-Response message is encrypted by the SKDC to the requesting service principal using the secret key that is shared between the SKDC and the service principal.

The GSK-Response message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "GSK-REP".
- SP's realm (sprealm): This the name of the realm, domain or zone in which the SP belongs in connection to this request.
- SP's identity (spname): This is the identity of the SP requesting the group-key in the previous GSK-REQ message.
- Receipt (receipt): The receipt carries the group-key and relevant parameters. It is encrypted by the SKDC to the SP using the secret key shared between the SKDC and the cSP. The receipt (receipt) contains the following:
  - Receipts flags - optional (rflags).
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Multicast group realm - optional (mcastrealm).
  - Multicast group identity (mcastname).
  - Group permissions - optional (grppac).
  - Current time (authtime).
  - Nonce from the GSK-REQ request (nonce).
  - Expiration time of key (endtime).

    * Group key (keydata): The key-data structure contains the group-key destined for the requesting SP. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.3. GSK-Fetch Message (GSK-FET)

The GSK-Fetch message is sent by a client to the SKDC asking for a copy of a group-key asociated with a multicast group. The client must identify the desired multicast group name (mcastname) in the SKDC-REQ-BODY of the message. The client authenticates itself to the SKDC by including an authenticator.

The contents of the GSK-Fetch message is as follows:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-FET".

- SKDC request body (req-body): The req-body contains the parameters required by the SKDC identify the multicast group. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a GSK-FET message contains the following:

    * SKDC options - optional (skdc-options).

    * SKDC’s realm - optional (skdcrealm).

    * SKDC’s identity (skdcname).

    * Multicast group realm - optional (mcastrealm).

    * Multicast group identity (spname): This is the name of the multicast group whose group-key is being fetched.

    * KeyID - optional (keyid).

- Authenticator (authenticator): The authenticator in the GSK-FET is encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information the the authenticator structure. The authenticator in the GSK-FET contains the following:

    * Client’s realm (crealm).

    * Client’s identity (cname).

    * Client’s current time (ctime).

    * A new nonce generated by the client (nonce).

    * Sequence number - optional (seqnum).

    * Checksum - optional (cksum).

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.4. GSK-Deliver Message (GSK-DLVR)

The GSK-Deliver (GSK-DLVR) message is sent from the SKDC to the client in response to the client’s GSK-Fetch message. The GSK-Deliver message uses the receipt structure to carry the group encryption key. The receipt is encrypted using secret key which is shared pair-wise between the client and the SKDC.

The contents of the GSK-Deliver message is as follows:

- Protocol version (pvno): This is the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-DLVR".

- Client’s realm (crealm): This the name of the realm, domain or group in which the client belongs in connection to this request.

- Client’s identity (cname): This is the identity of the client.

- Receipt (receipt): This is the part of the GSK-Deliver message carries the group-key. It is encrypted by the SKDC to the client using the secret key shared between the SKDC and the client. The receipt contains the following:

    * Receipts flags - optional (rflags).

    * Issuing SKDC’s realm (skdcrealm).

    * Issuing SKDC’s identity (skdcname).

    * Multicast group realm - optional (mcastrealm).

    * Multicast group identity (mcastname): This is the name of the multicast group whose group-key is being delivered.

  * Group permissions - optional (grppac).

  * Current time (authtime).

  * Nonce from the previous GSK-FET message (nonce).

  * Expiration time of key (endtime).

  * Group key (keydata): This key-data structure contains the group key destined for the requesting client. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

6. Group Shared Key Deletion

The current protocol supports the proactive removal of a group-key associated with a multicast group prior to the expiration of the group-key. Only the creator (owner) of the multicast group can request the SKDC to delete a group-key (or the SKDC itself could perform a group-key deletion in response to an external authorized trigger).

Due to the nature of a symmetric group-key, the removal of a group-key from a multicast group requires the SKDC to issue unicast PSK-Delete messages to each known member of the group. When the SKDC sends the PSK-Delete message to a client who is a group member, the SKDC must identify the group-key via its KeyID. Each group member must respond to the SKDC with a PSK-Delete-Confirm (PSK-DELC) message to confirm the deletion or erasure of the group-key. See Section Section 4.1 for more information on the PSK-DELT and PSK-DELC messages.

The SKDC must wait for all known group members to individually confirm (via a PSK-DELC message) their deletion of the group-key. Only then should the SKDC return a GSK-Delete-Confirmation (GSK-DELC) message to the service principal who requested the group-key deletion.

When the SKDC is in the process of deleting a group-key, it must deny further requests for the group-key from new members.

The group-key deletion process involves four types of messages:

- GSK-Delete (GSK-DELT): The SP (as group-owner) sends a GSK-Delete request to the SKDC.

- PSK-Delete (KeyID): For each member of the group (i.e. clients who previosuly requested a copy of the group-key), the SKDC sends a unicast PSK-Delete (PSK-DELT) message to that client identifying the target key to be deleted (by KeyID). See Section Section 4.1 for the PSK-DELT message.

- PSK-Delete-Confirm (KeyID): Each group member responds after key-deletion with a PSK-DELC message to the SKDC. See Section Section 4.2for the PSK-DELC message.

- GSK-Delete-Confirmed (GSK-DELC): The SKDC responds with a GSK-Delete-Confirmed message to the SP (as group-owner) once all copies of the group-key has been deleted at the group-members.

The message flow for GSK deletion is shown in Figure 6.

The GSK Deletion Message Components

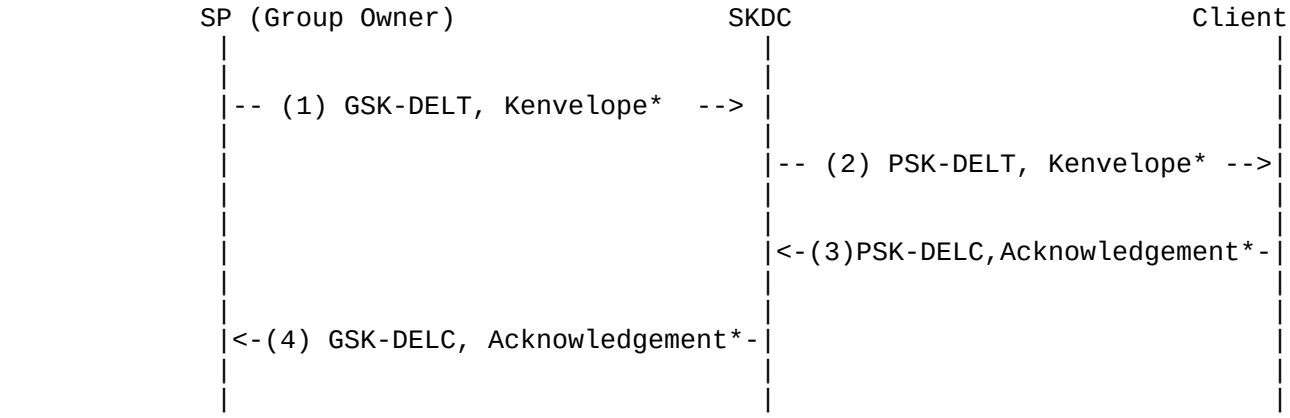


Figure 6

6.1. GSK-Delete Message (GSK-DELT)

The GSK-Delete (GSK-DELT) request message is sent from an SP (group owner) to the SKDC asking for the deletion or erasure of the group-key of the multicst group identified in the message.

The GSK-DELT message contains the following.

- Protocol version (pvno): This is the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-DELT".

- SP's realm (sprealm): This the name of the realm, domain or group in which the SP belongs in connection to this request.

- SP's identity (spname): This is the identity of the SP.

- Key Envelope (kenvelope): The key envelope is encrypted using the SP's secret key that it shares with the SKDC. The key envelope contains the following:

  * Envelope Flags - optional (envflags).

  * Multicast group realm - optional (mcastcrealm): This is the realm of the multicast group whose group-key is being deleted.

  * Multicast group identity (skdcname): This is the name of the multicast group whose group-key is being deleted.

  * Current time (authtime).

  * New nonce generated by the SP (nonce).

  * Sequence number - optional (seqnum).

  * Key data (keydata): This key-data structure contains the KeyID of the target key to be deleted. The key-data in a key envelope must never contain a cryptographic key. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

6.2. GSK-Delete-Confirm Message (GSK-DELC)

In response to a GSK-Delete request from a service principal (as group owner), the SKDC sends a GSK-Delete-Confirm (GSK-DELC) message to the service principal for the successful deletion of not only its copy of the group-key, but also the successful deletion of the copies of the group-key at each group member (client).

The GSK-DELC message contains the following:

- Protocol version (pvno): This is the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-DELC".

- SP's realm - optional (sprealm).
- SP's identity (spname).
- Acknowledgement (ack): The acknowledgement is encrypted by the SKDC to the SP using the secret key shared only between the SKDC and SP.
  - Multicast group identity (mcastname): This is the name of the multicast group whose group-key has been deleted.
  - Multicast group realm - optional (mcastrealm).
  - Nonce from the SP's previous GSK-DELT message (nonce).
  - Current time (authtime).
  - Sequence number - optional (seqnum): This field is optional.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

7. Public Key Pair Establishment

The current protocol supports the distribution of public key pairs and certificates. Since the SKDC is not a certificate authority (in the sense of RFC2459), issuing (signing) X509 certificates is out of scope for the current specification. If the SKDC receives from a client a request for a new certificate, the SKDC must obtain a certificate from a CA server (or CA service) located either in the same realm/zone or elsewhere on behalf of the requesting client. How the SKDC discovers CA services is out of scope for the current specification.

In the following we provide additional message types to support a client requesting the SKDC to deliver a new public key pair and to return the key pair to the client.

When a client seeks to obtain a new public key pair, the client sends a Public Key Pair Request (PKP-REQ) message to the SKDC. The requesting client must include an authenticator that is encrypted using the secret key it shares with the SKDC. The SKDC returns the key-pair in the receipt structure to the client (encrypted to the client).

As a further option to the PKP-REQ message, if the client identifies a service principal in the SKDC-REQ-BODY of the PKP-REQ message, the SKDC would also return a miniticket that contains only the public

half of the key-pair. The miniticket would be encrypted by the SKDC to the named service principal, using the secret key that the SKDC shares with the service principal. Note that this is a departure from the PSK-Request semantics (for symmetric key requests) because the miniticket contains a public key (instead of a symmetric key).

The client (or the SKDC) then sends the miniticket to the service principal who can decrypt the miniticket using the secret key it shares with the SKDC.

By encrypting the miniticket to the service principal, the SKDC is effectively attesting to the binding between the public key pair and the client's identity (without the use of digital certificates). The service principal trusts SKDC to provide this pseudo-attestation regarding the client as the owner of the new public key pair.

Note that in this approach the security of the public key pair is only as a secure as the symmetric key algorithm used to encrypt the receipt and the miniticket.

If additionally the client seeks to obtain a digital certificate for the new public key, then the client must indicate this option in the SKDC-Options field (in the SKDC-REQ-BODY) of the PKP-Request message. There are several options available related to digital certificates and CA certificates (trust anchors):

No certificate: This is the default setting for the PKP-Request message.

Certificate: This means the client additionally requests the return of a new X509 certificate corresponding to the new public key.

Include certificate chain: This option is meaningful only of the client requests a new X509 certificate. When set, this option means that the client also requests the certificate chain consisting of copies of all signing certificates to the top of the certificate hierarchy.

7.1. Public Key Pair Request (PKP-REQ)

A client uses the PKP-Request message (PKP-REQ) to request a new public key pair from the SKDC. The client must include an authenticator when sending this message to the SKDC. The PKP-REQ message contains the following:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "PKP-REQ".

- SKDC request body (req-body): The request body contains the parameters required by the SKDC in the context of this request. the See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a PKP-REQ message contains the following:

  * Key Type (kty): This is type of key being requested by the client from the SKDC.

  * Desired algorithm (etype): This is the algorithm that is desired (or supported) by the client.

  * SKDC options - optional (skdc-options): The client sets the "certificate request" option (and possibly the "certificate chain" option) in this field if it requests a digital certificate (and corresponding chain) to be returned together with the public key pair.

  * SKDC’s realm (skdcrealm).

  * SKDC’s identity (skdcname).

  * Service principal’s realm - optional (sprealm):

  * Service principal’s identity -- optional (spname):

    + spname is present: If the spname is present, this means that the client wishes for the SKDC to prepare copy of the new public key only for the named service principal via the miniticket structure.

    + spname is absent: If the spname is absent the SKDC delivers the public key pair only to the client via the receipt structure.

  * Client permissions - optional (cpac).

- Authenticator (authenticator): This is the authenticator encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information on the authenticator structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

7.2. Public Key Pair Response (PKP-REP)

In response to the PKP-Request message (PKP-REQ) from a client entity, the SKDC returns a copy of the new public key pair to the client in a receipt structure, encrypted using the secret key shared between the client and SKDC. This ensures that only the client is in possession of the new public key pair (notably the private key).

If the SKDC-REQ-BODY of the client's previous PKP-request message contains the identity of a service principal (spname), the SKDC must also create a miniticket containing a copy of the public key only. The miniticket is encrypted to the service principal.

The PKP-REP message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PKP-REP".
- Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.
- Client's identity (cname): This is the identity of the client.
- Miniticket - optional (mticket): If the client identified a service principal in the previous PKP-REQ message (in its SKDC-REG-BODY), then a miniticket is included by the SKDC in this PKP-REP message. If present, the miniticket contains the following:
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Service principal's realm - optional (sprealm).
  - Service Principal's identity (spname).
  - Encrypted miniticket part (enc-part): This is the part of the PKP-REP message that is encrypted by the SKDC to the service principal, using the secret key that the SKDC shares pair-wise with the service principal. It contains the following:
    - Ticket flags - optional (tflags).
    - Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.

+ Client’s identity (cname).

+ Client permissions - optional (cpac).

+ Time of authentication (authtime).

+ Expiration time of this key - optional (endtime): This field is present if the SKDC returns a public key pair only. If a certificate accompanies the public key pair, then this field is absent.

+ Service principal permissions - optional (sppac).

+ Transited realms - optional (transited).

+ Key data (keydata): In a PKP-REP message, the key-data structure in the miniticket contains only the public key of the client. The private key must not be included. If the client had also requested a new certificate, the certificate is included here. See section Section 2.6 for the key-data structure.

o Receipt (receipt): This is the part of the PKP-REP message that is encrypted by the SKDC to the client, using the secret key that the SKDC shares pair-wise with the client. It contains the following:

* Receipts flags - optional (tflags).

* Issuing SKDC’s realm (skdcrealm).

* Issuing SKDC’s identity (skdcname).

* Service principal’s realm - optional (sprealm).

* Service Principal’s identity (spname).

* Service Principal’s SKDC - optional (spskdc).

* Client permissions - optional (cpac).

* Time of authentication (authtime): This is the time of the creation of this receipt.

* Nonce from the Client’s previous PSK-REQ request message (nonce).

* Expiration time of this key (endtime).

* Key data (keydata): In a PKP-REP message, the key-data structure in the receipt contains both the public key and private key belonging to the requesting client. If the client had also requested a new certificate, the certificate is included here. See section Section 2.6 for the key-data structure.

o Extensions - optional (ext): Reserved for future extensions. This field is optional.

8. JSON Message Format

TBD.

9. Encryption and Checksums

TBD.

10. Security Considerations

TBD.

11. Privacy Considerations

TBD.

12. IANA Considerations

TBD.

13. Acknowledgments

We thank Jesse Walker for design inputs and initial review.

14. References

14.1. Normative References

[JSON] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", March 2014, <https://tools.ietf.org/html/rfc7159>.

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

[RFC6347] Rescorla, E., "Datagram Transport Layer Security Version 1.2", January 2012, <http://tools.ietf.org/html/rfc6347>.

[RFC7252] Shelby, Z., "The Constrained Application Protocol (CoAP)", June 2014, <http://tools.ietf.org/html/rfc7252>.

14.2. Informative References

[ACE] Seitz, L., Ed., "ACE Use Cases", October 2012, <https://tools.ietf.org/wg/ace/draft-ietf-ace-usecases/>.

[BR-3KPD] Bellare, M. and P. Rogaway, "Entity Authentication and Key Distribution (In Advances in Cryptology, pages 110-125. Springer-Verlag, 1993)", September 1993, <http://link.springer.com/>.

[Choo04] Choo, K., Boyd, C., Hitchcock, Y., and G. Maitland, "On Session Identifiers in Provably Secure Protocols (Security in Communication Networks 4th International Conference, SCN 2004)", September 2004, <http://link.springer.com/>.

[Choo06] Choo, R., "Key Establishment: Proofs and Refutations", May 2006, <http://eprints.qut.edu.au/16262/1/Kim-Kwang_Choo_Thesis.pdf>.

[EPID] Brickell, E. and J. Li, "Enhanced Privacy ID (in NIST Privacy Enhancing Cryptography Conference 2011)", December 2011, <http://csrc.nist.gov/groups/ST/PEC2011/presentations2011/brickell.pdf>.

[MSKILE] Microsoft, ., "Kerberos Protocol Extensions (v20140502)", May 2014, <https://msdn.microsoft.com/en-us/library/cc233855.aspx>.

[MSPAC] Microsoft, ., "Privilege Attribute Certificate Data Structure (v20140502)", May 2014, <https://msdn.microsoft.com/en-us/library/cc237917.aspx>.

[NS] Needham, R. and M. Schroeder, "Using encryption for authentication in large networks of computers (CACM)", December 1978, <http://en.wikipedia.org/wiki/Needham-Schroeder_protocol>.

[OAuth2] Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, <http://tools.ietf.org/html/rfc6749>.

[OIDC] Sakimura, N., "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos V5", February 2005, <http://tools.ietf.org/html/rfc3961>.

[RFC3986] Berners-Lee, T., "Uniform Resource Identifier (URI): Generic Syntax", January 2005, <http://www.ietf.org/rfc/rfc3986.txt>.

[RFC4120] Neuman, C., "The Kerberos Network Authentication Service (V5)", July 2005, <http://tools.ietf.org/html/rfc4120>.

[RFC6113] Hartman, S., "A Generalized Framework for Kerberos Pre-Authentication", April 2011, <http://tools.ietf.org/html/rfc6113>.

[TPM] TCG, ., "Trusted Platform Module (TPM) Main Specification Level 2 Version 1.2", March 2011, <http://www.trustedcomputinggroup.org/resources/tpm_main_specification>.

[UMACORE] Hardjono, T., Ed., "User-Managed Access (UMA) Profile of OAuth 2.0", November 2014, <https://docs.kantarainitiative.org/uma/draft-uma-core.html>.

Appendix A. Document History

NOTE: To be removed by RFC editor before publication as an RFC.

Authors' Addresses

Thomas Hardjono
MIT

Email: hardjono@mit.edu

Ned Smith
Intel Corp

Email: ned.smith@intel.com

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

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

October 22, 2018

An architecture for authorization in constrained environments
draft-ietf-ace-actors-07

Abstract

Constrained-node networks are networks where some nodes have severe constraints on code size, state memory, processing capabilities, user interface, power and communication bandwidth (RFC 7228).

This document provides terminology, and identifies the elements that an architecture needs to address, providing a problem statement, for authentication and authorization in these networks.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

Table of Contents

1. Introduction

As described in [RFC7228], constrained nodes are small devices with limited abilities which in many cases are made to fulfill a specific simple task. They may have limited hardware resources such as processing power, memory, non-volatile storage and transmission capacity and additionally in most cases do not have user interfaces and displays. Due to these constraints, commonly used security protocols are not always easily applicable, or may give rise to particular deployment/management challenges.

As components of the Internet of Things (IoT), constrained nodes are expected to be integrated in all aspects of everyday life and thus will be entrusted with vast amounts of data. Without appropriate security mechanisms attackers might gain control over things relevant to our lives. Authentication and authorization mechanisms are therefore prerequisites for a secure Internet of Things.

Applications generally require some degree of authentication and authorization, which gives rise to some complexity. Authorization is about who can do what to which objects (see also [RFC4949]). Authentication specifically addresses the who, but is often specific to the authorization that is required (for example, it may be sufficient to authenticate the age of an actor, so no identifier is needed or even desired). Authentication often involves credentials, only some of which need to be long-lived and generic; others may be directed towards specific authorizations (but still possibly long-lived). Authorization then makes use of these credentials, as well as other information (such as the time of day). This means that the complexity of authenticated authorization can often be moved back and forth between these two aspects.

In some cases authentication and authorization can be addressed by static configuration provisioned during manufacturing or deployment by means of fixed trust anchors and static access control lists. This is particularly applicable to siloed, fixed-purpose deployments.

However, as the need for flexible access to assets already deployed increases, the legitimate set of authorized entities as well as their specific privileges cannot be conclusively defined during deployment, without any need for change during the lifetime of the device. Moreover, several use cases illustrate the need for fine-grained access control policies, for which for instance a basic access control list concept may not be sufficiently powerful [RFC7744].

The limitations of the constrained nodes impose a need for security mechanisms which take the special characteristics of constrained environments into account; not all constituents may be able to

perform all necessary tasks by themselves. To put it the other way round: the security mechanisms that protect constrained nodes must remain effective and manageable despite the limitations imposed by the constrained environment.

Therefore, in order to be able to achieve complex security objectives between actors some of which are hosted on simple ("constrained") devices, some of the actors will make use of help from other, less constrained actors. (This offloading is not specific to networks with constrained nodes, but their constrainedness as the main motivation is.)

We therefore group the logical functional entities by whether they can be assigned to a constrained device ("constrained level") or need higher function platforms ("less-constrained level"); the latter does not necessarily mean high-function, "server" or "cloud" platforms. Note that assigning a logical functional entity to the constrained level does not mean that the specific implementation needs to be constrained, only that it _can_ be.

The description assumes that some form of setup (aspects of which are often called provisioning and/or commissioning) has already been performed and at least some initial security relationships important for making the system operational have already been established.

This document provides some terminology, and identifies the elements an architecture needs to address, representing the relationships between the logical functional entities involved; on this basis, a problem description for authentication and authorization in constrained-node networks is provided.

1.1. Terminology

Readers are assumed to be familiar with the terms and concepts defined in [RFC4949], including "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

REST terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252]; the latter also defines additional terms such as "endpoint".

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. is defined in [RFC7228].

In addition, this document uses the following terminology:

Resource (R): an item of interest which is represented through an interface. It might contain sensor or actuator values or other information. (Intended to coincide with the definitions of [RFC7252] and [RFC7231].)

Constrained node: a constrained device in the sense of [RFC7228].

Actor: A logical functional entity that performs one or more tasks. Multiple actors may be present within a single device or a single piece of software.

Resource Server (RS): An entity which hosts and represents a Resource. (Used here to discuss the server that provides a resource that is the end, not the means, of the authenticated authorization process – i.e., not CAS or AS.)

Client (C): An entity which attempts to access a resource on a RS. (Used to discuss the client whose access to a resource is the end, not the means, of the authenticated authorization process.)

Overseeing principal: (Used in its English sense here, and specifically as:) An individual that is either RqP or RO or both.

Resource Owner (RO): The overseeing principal that is in charge of the resource and controls its access permissions.

Requesting Party (RqP): The overseeing principal that is in charge of the Client and controls the requests a Client makes and its acceptance of responses.

Authorization Server (AS): An entity that prepares and endorses authentication and authorization data for a Resource Server.

Client Authorization Server (CAS): An entity that prepares and endorses authentication and authorization data for a Client.

Authorization Manager: An entity that prepares and endorses authentication and authorization data for a constrained node. Used in constructions such as "a constrained node’s authorization manager" to denote AS for RS and CAS for C.

Authenticated Authorization: The confluence of mechanisms for authentication and authorization, ensuring that authorization is applied to and made available for authenticated entities and that entities providing authentication services are authorized to do so for the specific authorization process at hand.

Note that other authorization architectures such as OAuth [RFC6749] or UMA [I-D.hardjono-oauth-umacore] focus on the authorization problems on the RS side, in particular what accesses to resources the RS is to allow. In this document the term authorization includes this aspect, but is also used for the client-side aspect of authorization, i.e., more generally allowing RqPs to decide what interactions clients may perform with other endpoints.

2. Architecture and High-level Problem Statement

This document deals with how to control and protect resource-based interaction between potentially constrained endpoints. The following setting is assumed as a high-level problem statement:

- An endpoint may host functionality of one or more actors.
- C in one endpoint requests to access R on a RS in another endpoint.
- A priori, the endpoints do not necessarily have a pre-existing security relationship to each other.
- Either of the endpoints, or both, may be constrained.

2.1. Elements of an Architecture

In its simplest expression, the architecture starts with a two-layer model: the principal level (at which components are assumed to be functionally unconstrained) and the constrained level (at which some functional constraints are assumed to apply to the components).

Without loss of generality, we focus on the C functionality in one endpoint, which we therefore also call C, accessing the RS functionality in another endpoint, which we therefore also call RS.

The constrained level and its security objectives are detailed in Section 5.1.

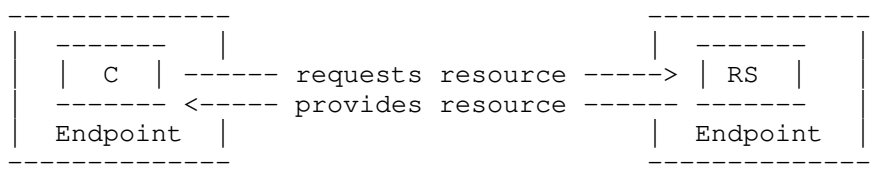


Figure 1: Constrained Level

The authorization decisions at the endpoints are made on behalf of the overseeing principals that control the endpoints. To reuse OAuth and UMA terminology, the present document calls the overseeing principal that is controlling C the Requesting Party (RqP), and calls the overseeing principal that is controlling RS the Resource Owner (RO). Each overseeing principal makes authorization decisions (possibly encapsulating them into security policies) which are then enforced by the endpoint it controls.

The specific security objectives will vary, but for any specific version of this scenario will include one or more of:

- Objectives of type 1: No entity not authorized by the RO has access to (or otherwise gains knowledge of) R.

- Objectives of type 2: C is exchanging information with (sending a request to, accepting a response from) a resource only where it can ascertain that RqP has authorized the exchange with R.

Objectives of type 1 require performing authorization on the Resource Server side while objectives of type 2 require performing authorization on the Client side.

More on the security objectives of the principal level in Section 5.2.

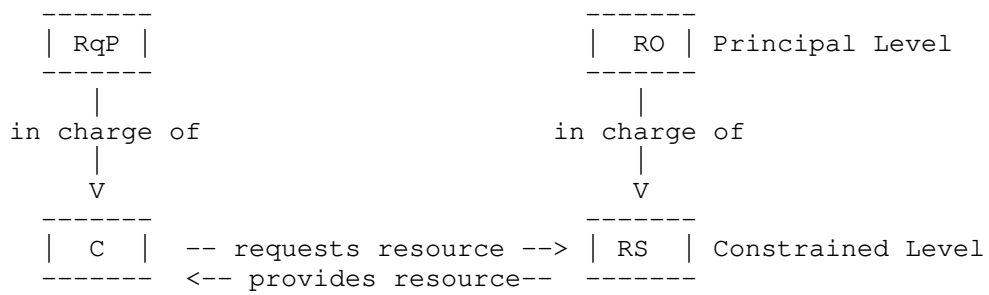


Figure 2: Constrained Level and Principal Level

The use cases defined in [RFC7744] demonstrate that constrained devices are often used for scenarios where their overseeing principals are not present at the time of the communication, are not able to communicate directly with the device because of a lack of user interfaces or displays, or may prefer the device to communicate autonomously.

Moreover, constrained endpoints may need support with tasks requiring heavy processing, large memory or storage, or interfacing to humans,

such as management of security policies defined by an overseeing principal. The principal, in turn, requires some agent maintaining the policies governing how its endpoints will interact.

For these reasons, another level of nodes is introduced in the architecture, the less-constrained level (illustrated below in Figure 3). Using OAuth terminology, AS acts on behalf of the RO to control and support the RS in handling access requests, employing a pre-existing security relationship with RS. We complement this with CAS acting on behalf of RqP to control and support the C in making resource requests and acting on the responses received, employing a pre-existing security relationship with C. To further relieve the constrained level, authorization (and related authentication) mechanisms may be employed between CAS and AS (Section 6.2). (Again, both CAS and AS are conceptual entities controlled by their respective overseeing principals. Many of these entities, often acting for different overseeing principals, can be combined into a single server implementation; this of course requires proper segregation of the control information provided by each overseeing principal.)

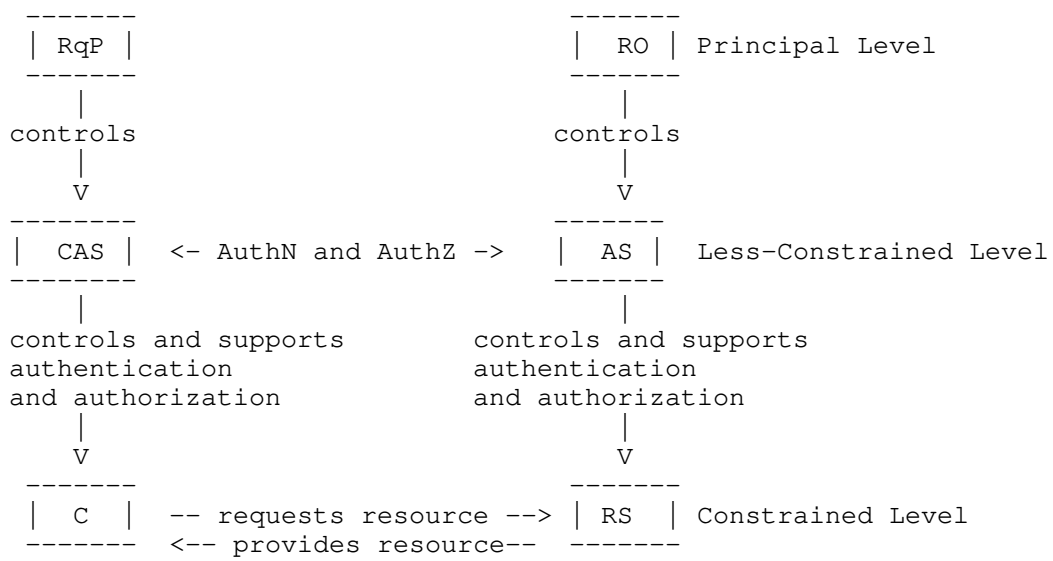


Figure 3: Overall architecture

Figure 3 shows all three levels considered in this document. Note that the vertical arrows point down to illustrate exerting control and providing support; this is complemented by information flows that often are bidirectional. Note also that not all entities need to be ready to communicate at any point in time; for instance, RqP may have

provided enough information to CAS that CAS can autonomously negotiate access to RS with AS for C based on this information.

2.2. Architecture Variants

The elements of the architecture described above are indeed architectural; that is, they are parts of a conceptual model, and may be instantiated in various ways in practice. For example, in a given scenario, several elements might share a single device or even be combined in a single piece of software. If C is located on a more powerful device, it can be combined with CAS:

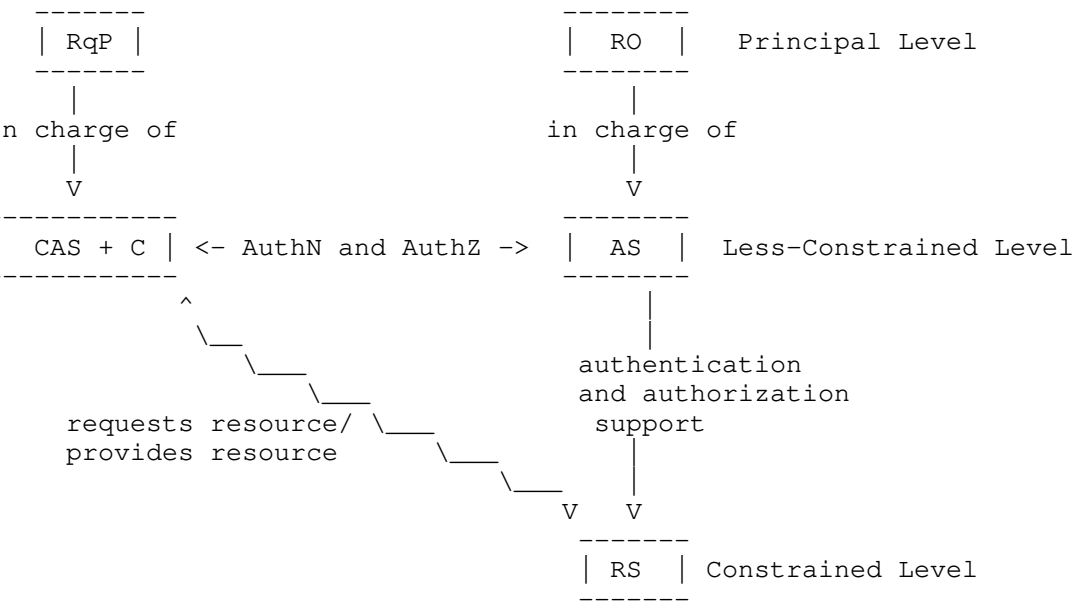


Figure 4: Combined C and CAS

If RS is located on a more powerful device, it can be combined with AS:

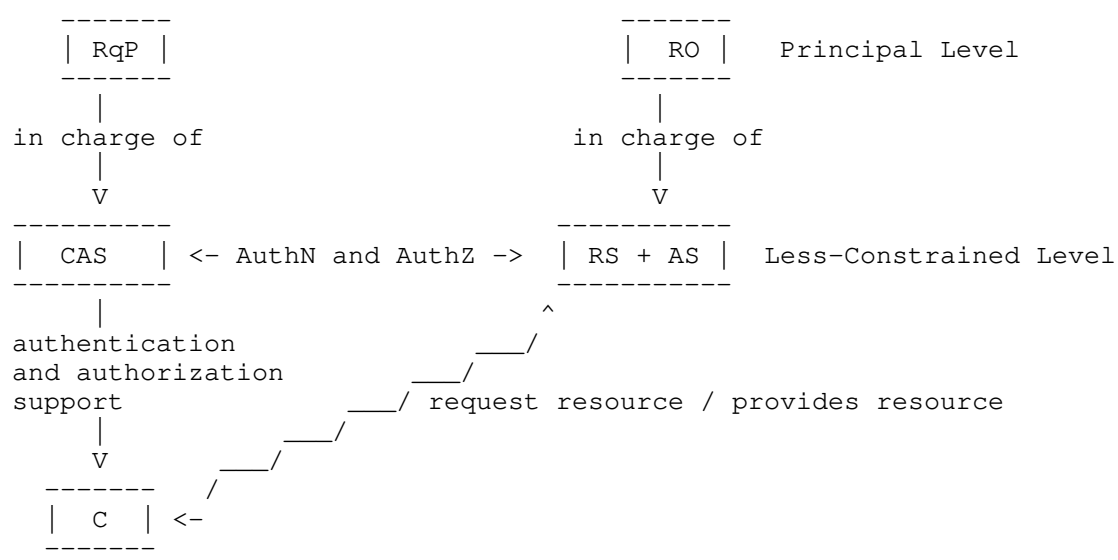


Figure 5: Combined AS and RS

If C and RS have the same overseeing principal, CAS and AS can be combined.

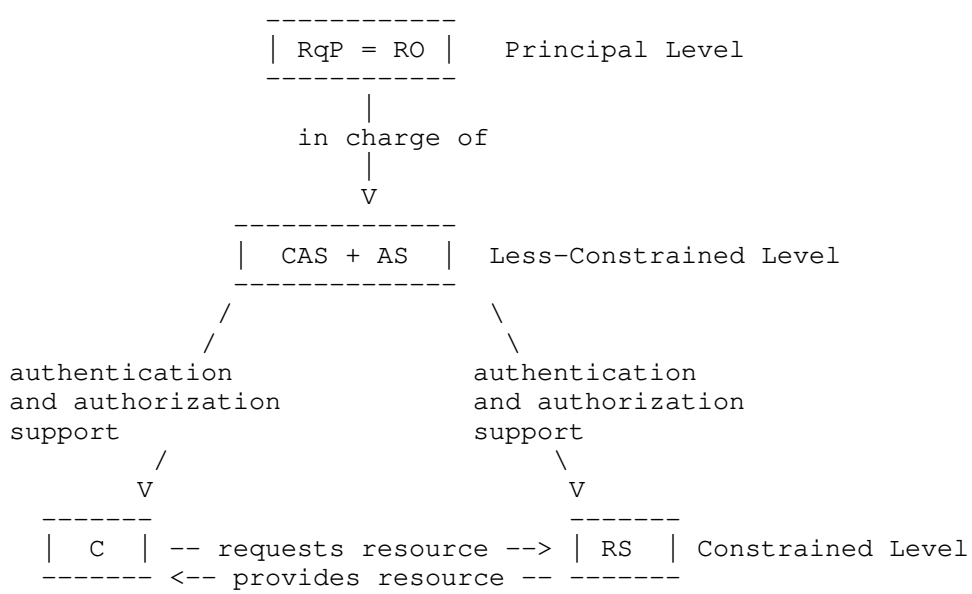


Figure 6: CAS combined with AS

2.3. Information Flows

We now formulate the problem statement in terms of the information flows the architecture focuses on. (While the previous section discusses the architecture in terms of abstract devices and their varying roles, the actual protocols being standardized define those information flows and the messages embodying them: "RESTful architectures focus on defining interfaces and not components" ([REST], p. 116).)

The interaction with the nodes on the principal level, RO and RqP, is not involving constrained nodes and therefore can employ an existing mechanism. The less-constrained nodes, CAS and AS, support the constrained nodes, C and RS, with control information, for example permissions of clients, conditions on resources, attributes of client and resource servers, keys and credentials. This control information may be rather different for C and RS.

The potential information flows are shown in Figure 7. The direction of the vertical arrows expresses the exertion of control; actual information flow is bidirectional.

The message flow may pass unprotected paths and thus need to be protected, potentially beyond a single REST hop (Section 3.1):

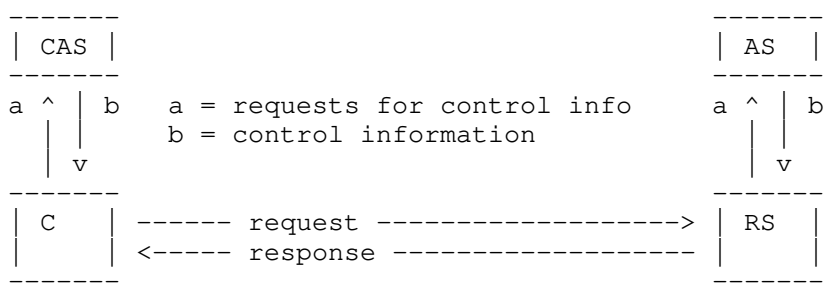


Figure 7: Information flows that need to be protected

- We assume that the necessary keys/credentials for protecting the control information between the potentially constrained nodes and their associated less-constrained nodes are pre-established, for example as part of the commissioning procedure.

- Any necessary keys/credentials for protecting the interaction between the potentially constrained nodes will need to be established and maintained as part of a solution.

In terms of the elements of the architecture laid out above, this document's problem statement for authorization in constrained environments can then be summarized as follows:

- The interaction between potentially constrained endpoints is controlled by control information provided by less-constrained nodes on behalf of the overseeing principals of the endpoints.

- The interaction between the endpoints needs to be secured, as well as the establishment of the necessary keys for securing the interaction, potentially end-to-end through intermediary nodes.

- The mechanism for transferring control information needs to be secured, potentially end-to-end through intermediary nodes. Pre-established keying material may need to be employed for establishing the keys used to protect these information flows.

(Note that other aspects relevant to secure constrained node communication such as secure bootstrap or group communication are not specifically addressed by the present document.)

3. Security Objectives

The security objectives that are addressed by an authorization solution include confidentiality and integrity. Additionally, an authorization solution has an impact on the availability: First, by reducing the load (only accepting selected operations by selected entities limits the burden on system resources), and second, because misconfigured or wrongly designed authorization solutions can result in availability breaches (denial of service) as users might no longer be able to use data and services as they are supposed to.

Authentication mechanisms can help achieve additional security objectives such as accountability and third-party verifiability. These additional objectives are not directly related to authorization and thus are not in scope of this draft, but may nevertheless be relevant. Accountability and third-party verifiability may require authentication on a device level, if it is necessary to determine which device performed an action. In other cases it may be more important to find out who is responsible for the device's actions. (The ensuing requirements for logging, auditability, and the related integrity requirements are very relevant for constrained devices as well, but outside the scope of this document.) See also Section 4 for more discussion about authentication and authorization.

The security objectives and their relative importance differ for the various constrained environment applications and use cases [RFC7744].

The architecture is based on the observation that different parties may have different security objectives. There may also be a "collaborative" dimension: to achieve a security objective of one party, another party may be required to provide a service. For example, if RqP requires the integrity of representations of a resource R that RS is hosting, both C and RS need to partake in integrity-protecting the transmitted data. Moreover, RS needs to protect any write access to this resource as well as to relevant other resources (such as configuration information, firmware update resources) to prevent unauthorized users from manipulating R.

3.1. End-to-End Security Objectives in Multi-Hop Scenarios

In many cases, the information flows described in Section 2.3 cross multiple client-server pairings but still need to be protected end-to-end. For example, AS may not be connected to RS (or may not want to exercise such a connection), relying on C for transferring authorization information. As the authorization information is related to the permissions granted to C, C must not be in a position to manipulate this information, which therefore requires integrity protection on the way between AS and RS.

As another example, resource representations sent between endpoints may be stored in intermediary nodes, such as caching proxies or pub-sub brokers. Where these intermediaries cannot be relied on to fulfill the security objectives of the endpoints, it is the endpoints that will need to protect the exchanges beyond a single client-server exchange.

Note that there may also be cases of intermediary nodes that very much partake in the security objectives to be achieved. The question what are the pairs of endpoints between which the communication needs end-to-end protection (and which aspect of protection) is defined by the specific use case. Two examples of intermediary nodes executing security functionality:

- To enable a trustworthy publication service, a pub-sub broker may be untrusted with the plaintext content of a publication (confidentiality), but required to verify that the publication is performed by claimed publisher and is not a replay of an old publication (authenticity/integrity).

- To comply with requirements of transparency, a gateway may be allowed to read, verify (authenticity) but not modify (integrity) a resource representation which therefore also is end-to-end integrity protected from the server towards a client behind the gateway.

In order to support the required communication and application security, keying material needs to be established between the relevant nodes in the architecture.

4. Authentication and Authorization

Server-side authorization solutions aim at protecting the access to items of interest, for instance hardware or software resources or data: They enable the resource owner to control who can access it and how.

To determine if an entity is authorized to access a resource, an authentication mechanism is needed. According to the Internet Security Glossary [RFC4949], authentication is "the process of verifying a claim that a system entity or system resource has a certain attribute value." Examples for attribute values are the ID of a device, the type of the device or the name of its owner.

The security objectives the authorization mechanism aims at can only be achieved if the authentication and the authorization mechanism work together correctly. We speak of authenticated authorization to

refer to the required synthesis of mechanisms for authentication and authorization.

Where used for authorization, the set of authenticated attributes must be meaningful for this purpose, i.e., authorization decisions must be possible based on these attributes. If the authorization policy assigns permissions to an individual entity, the set of authenticated attributes must be suitable to uniquely identify this entity.

In scenarios where devices are communicating autonomously there is often less need to uniquely identify an individual device: For an overseeing principal, the fact that a device belongs to a certain company or that it has a specific type (such as a light bulb) or location may be more important than that it has a unique identifier.

Overseeing principals (RqP and RO) need to decide about the required level of granularity for the authorization. For example, we distinguish device authorization from owner authorization, and binary authorization from unrestricted authorization. In the first case different access permissions are granted to individual devices while in the second case individual owners are authorized. If binary authorization is used, all authenticated entities are implicitly authorized and have the same access permissions. Unrestricted authorization for an item of interest means that no authorization mechanism is used for accessing this resource (not even by authentication) and all entities are able to access the item as they see fit (note that an authorization mechanism may still be used to arrive at the decision to employ unrestricted authorization).

Authorization granularity	Authorization is contingent on:
device	authentication of specific device
owner	(authenticated) authorization by owner
binary	(any) authentication
unrestricted	(unrestricted access; access always authorized)

Table 1: Some granularity levels for authorization

More fine-grained authorization does not necessarily provide more security but can be more flexible. Overseeing principals need to consider that an entity should only be granted the permissions it

really needs (principle of least privilege), to ensure the confidentiality and integrity of resources.

Client-side authorization solutions aim at protecting the client from disclosing information to or ingesting information from resource servers RqP does not want it to interact with in the given way. Again, binary authorization (the server can be authenticated) may be sufficient, or more fine-grained authorization may be required. The client-side authorization also pertains to the level of protection required for the exchanges with the server (e.g., confidentiality). In the browser web, client-side authorization is often left to the human user that directly controls the client; a constrained client may not have that available all the time but still needs to implement the wishes of the overseeing principal controlling it, the RqP.

For the cases where an authorization solution is needed (all but unrestricted authorization), the enforcing party needs to be able to authenticate the party that is to be authorized. Authentication is therefore required for messages that contain (or otherwise update) representations of an accessed item. More precisely: The enforcing party needs to make sure that the receiver of a message containing a representation is authorized to receive it, both in the case of a client sending a representation to a server and vice versa. In addition, it needs to ensure that the actual sender of a message containing a representation is indeed the one authorized to send this message, again for both the client-to-server and server-to-client case. To achieve this, integrity protection of these messages is required: Authenticity of the message cannot be assured if it is possible for an attacker to modify it during transmission.

In some cases, only one side (client or server side) requires the integrity and / or confidentiality of a resource value. Overseeing principals may decide to omit authentication (unrestricted authorization), or use binary authorization (just employing an authentication mechanism). However, as indicated in Section 3, the security objectives of both sides must be considered, which can often only be achieved when the other side can be relied on to perform some security service.

5. Actors and their Tasks

This and the following section look at the resulting architecture from two different perspectives: This section provides a more detailed description of the various "actors" in the architecture, the logical functional entities performing the tasks required. The following section then will focus on the protocols run between these functional entities.

For the purposes of this document, an actor consists of a set of tasks and additionally has a security domain (client domain or server domain) and a level (constrained, principal, less-constrained). Tasks are assigned to actors according to their security domain and required level.

Note that actors are a concept to understand the security requirements for constrained devices. The architecture of an actual solution might differ as long as the security requirements that derive from the relationship between the identified actors are considered. Several actors might share a single device or even be combined in a single piece of software. Interfaces between actors may be realized as protocols or be internal to such a piece of software.

5.1. Constrained Level Actors

As described in the problem statement (see Section 2), either C or RS or both of them may be located on a constrained node. We therefore define that C and RS must be able to perform their tasks even if they are located on a constrained node. Thus, C and RS are considered to be Constrained Level Actors.

C performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including access requests.
- Validate that the RqP ("client-side") authorization information allows C to communicate with RS as a server for R (i.e., from C's point of view, RS is authorized as a server for the specific access to R).

RS performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including responses to access requests.
- Validate that the RO ("server-side") authorization information allows RS to grant C access to the requested resource as requested (i.e., from RS' point of view, C is authorized as a client for the specific access to R).

R is an item of interest such as a sensor or actuator value. R is considered to be part of RS and not a separate actor. The device on which RS is located might contain several resources controlled by different ROs. For simplicity of exposition, these resources are described as if they had separate RS.

As C and RS do not necessarily know each other they might belong to different security domains.

(See Figure 8.)

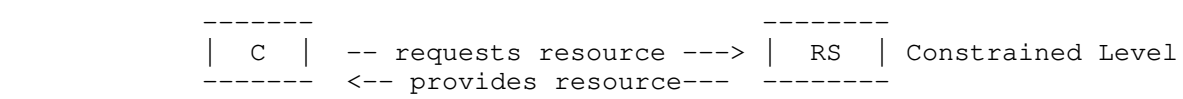


Figure 8: Constrained Level Actors

5.2. Principal Level Actors

Our objective is that C and RS are under control of overseeing principals in the physical world, the Requesting Party (RqP) and the Resource Owner (RO) respectively. The overseeing principals decide about the security policies of their respective endpoints; each overseeing principal belongs to the same security domain as their endpoints.

RqP is in charge of C, i.e. RqP specifies security policies for C, such as with whom C is allowed to communicate. By definition, C and RqP belong to the same security domain.

RqP must fulfill the following task:

- Configure for C authorization information for sources for R.

RO is in charge of R and RS. RO specifies authorization policies for R and decides with whom RS is allowed to communicate. By definition, R, RS and RO belong to the same security domain.

RO must fulfill the following task:

- Configure for RS authorization information for accessing R.

(See Figure 2.)

5.3. Less-Constrained Level Actors

Constrained level actors can only fulfill a limited number of tasks and may not have network connectivity all the time. To relieve them from having to manage keys for numerous endpoints and conducting computationally intensive tasks, another level of complexity for actors is introduced (and, thus, a stricter limit on their constrainedness). An actor on the less-constrained level belongs to

the same security domain as its respective constrained level actor. They also have the same overseeing principal.

The Client Authorization Server (CAS) belongs to the same security domain as C and RqP. CAS acts on behalf of RqP. It assists C in authenticating RS and determining if RS is an authorized server for R. CAS can do that because for C, CAS is the authority for claims about RS.

CAS performs the following tasks:

- Vouch for the attributes of its clients.
- Ascertain that C's overseeing principal (RqP) authorized AS to vouch for RS and provide keying material for it.
- Provide revocation information concerning its clients (optional).
- Obtain authorization information about RS from C's overseeing principal (RqP) and provide it to C.
- Negotiate means for secure communication to communicate with C.

The Authorization Server (AS) belongs to the same security domain as R, RS and RO. AS acts on behalf of RO. It supports RS by authenticating C and determining C's permissions on R. AS can do that because for RS, AS is the authority for claims about C.

AS performs the following tasks:

- Vouch for the attributes of its resource servers.
- Ascertain that RS's overseeing principal (RO) authorized CAS to vouch for C and provide keying material for it.
- Provide revocation information concerning its servers (optional).
- Obtain authorization information about C from RS' overseeing principal (RO) and provide it to RS.
- Negotiate means for secure communication to communicate with RS.

6. Kinds of Protocols

Devices on the less-constrained level potentially are more powerful than constrained level devices in terms of processing power, memory, non-volatile storage. This results in different characteristics for the protocols used on these levels.

6.1. Constrained Level Protocols

A protocol is considered to be on the constrained level if it is used between the actors C and RS which are considered to be constrained (see Section 5.1). C and RS might not belong to the same security domain. Therefore, constrained level protocols need to work between different security domains.

Commonly used Internet protocols can not in every case be applied to constrained environments. In some cases, tweaking and profiling is required. In other cases it is beneficial to define new protocols which were designed with the special characteristics of constrained environments in mind.

On the constrained level, protocols need to address the specific requirements of constrained environments. Examples for protocols that consider these requirements is the transfer protocol CoAP (Constrained Application Protocol) [RFC7252] and the Datagram Transport Layer Security Protocol (DTLS) [RFC6347] which can be used for channel security.

Constrained devices have only limited storage space and thus cannot store large numbers of keys. This is especially important because constrained networks are expected to consist of thousands of nodes. Protocols on the constrained level should keep this limitation in mind.

6.1.1. Cross Level Support Protocols

We refer to protocols that operate between a constrained device and its corresponding less-constrained device as cross-level support protocols. Protocols used between C and CAS or RS and AS are therefore support protocols.

Support protocols must consider the limitations of their constrained endpoint and therefore belong to the constrained level protocols.

6.2. Less-Constrained Level Protocols

A protocol is considered to be on the less-constrained level if it is used between the actors CAS and AS. CAS and AS might belong to different security domains.

On the less-constrained level, HTTP [RFC7230] and Transport Layer Security (TLS) [RFC8246] can be used alongside or instead of CoAP and DTLS. Moreover, existing security solutions for authentication and authorization such as the OAuth web authorization framework [RFC6749] and Kerberos [RFC4120] can likely be used without modifications and

the less-constrained layer is assumed to impose no constraints that would inhibit the traditional deployment/use of, e.g., a Public Key Infrastructure (PKI).

7. Elements of a Solution

Without anticipating specific solutions, the following considerations may be helpful in discussing them.

7.1. Authorization

The core problem we are trying to solve is authorization. The following problems related to authorization need to be addressed:

- AS needs to transfer authorization information to RS and CAS needs to transfer authorization information to C.
- The transferred authorization information needs to follow a defined format and encoding, which must be efficient for constrained devices, considering size of authorization information and parser complexity.
- C and RS need to be able to verify the authenticity of the authorization information they receive. C must ascertain that the authorization information stems from a CAS that was authorized by RqP, RS must validate that the authorization information stems from an AS that was authorized by RO.
- Some applications may require the confidentiality of authorization information. It then needs to be encrypted between CAS and C and AS and RS, respectively.
- C and RS must be able to check the freshness of the authorization information and determine for how long it is supposed to be valid.
- The RS needs to enforce the authorization decisions of the AS, while C needs to abide with the authorization decisions of the CAS. The authorization information might require additional policy evaluation (such as matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the constrained actors needs to be adapted to the capabilities of the devices implementing them.
- Finally, as is indicated in the previous bullet, for a particular authorization decision there may be different kinds of authorization information needed, and these pieces of information may be transferred to C and RS at different times and in different ways prior to or during the client request.

7.2. Authentication

The following problems need to be addressed, when considering authentication:

- RS needs to authenticate AS in the sense that it must be certain that it communicates with an AS that was authorized by RO, C needs to authenticate CAS in the sense that it must be certain that it communicates with a CAS that was authorized by RqP, to ensure that the authorization information and related data comes from the correct source.

- C must securely have obtained keying material to communicate with its CAS that is up to date and that is updated if necessary. RS must securely have obtained keying material to communicate with AS that is up to date and that is updated if necessary.

- CAS and AS may need to authenticate each other, both to perform the required business logic and to ensure that CAS gets security information related to the resources from the right source.

- In some use cases RS needs to authenticate some property of C, in order to map it to the relevant authorization information.

- C may need to authenticate RS, in order to ensure that it is interacting with the right resources.

- CAS and AS need to authenticate their communication partner (C or RS), in order to ensure it serves the correct device. If C and AS vouch for keying material or certain attributes of their respective constrained devices, they must ascertain that the devices actually currently have this keying material or these attributes.

7.3. Communication Security

There are different alternatives to provide communication security, and the problem here is to choose the optimal one for each scenario. We list the available alternatives:

- Session-based security at transport layer such as DTLS [RFC6347] offers security, including integrity and confidentiality protection, for the whole application layer exchange. However, DTLS may not provide end-to-end security over multiple hops. Another problem with DTLS is the cost of the handshake protocol, which may be too expensive for constrained devices especially in terms of memory and power consumption for message transmissions.

- An alternative is object security at application layer, for instance using [I-D.ietf-core-object-security]. Secure objects can be stored or cached in network nodes and provide security for a more flexible communication model such as publish/subscribe (compare e.g. CoRE Mirror Server [I-D.ietf-core-coap-pubsub]). A problem with object security is that it can not provide confidentiality for the message headers.

- Hybrid solutions using both session-based and object security are also possible. An example of a hybrid is where authorization information and cryptographic keys are provided by AS in the format of secure data objects, but where the resource access is protected by session-based security.

7.4. Cryptographic Keys

With respect to cryptographic keys, we see the following problems that need to be addressed:

Symmetric vs Asymmetric Keys
: We need keys both for protection of resource access and for protection of transport of authentication and authorization information. It may be necessary to support solutions that require the use of asymmetric keys as well as ones that get by with symmetric keys, in both cases. There are classes of devices that can easily perform symmetric cryptography, but consume considerably more time/battery for asymmetric operations. On the other hand asymmetric cryptography has benefits such as in terms of deployment.

Key Establishment
: How are the corresponding cryptographic keys established? Considering Section 7.1 there must be a mapping between these keys and the authorization information, at least in the sense that AS must be able to specify a unique client identifier which RS can verify (using an associated key). One of the use cases of [RFC7744] describes spontaneous change of access policies - such as giving a hitherto unknown client the right to temporarily unlock your house door. In this case C is not previously known to RS and a key must be provisioned by AS.

Revocation and Expiration
: How are keys replaced and how is a key that has been compromised revoked in a manner that reaches all affected parties, also keeping in mind scenarios with intermittent connectivity?

8. Assumptions and Requirements

In this section we list a set of candidate assumptions and requirements to make the problem description in the previous sections more concise and precise. Note that many of these assumptions and requirements are targeting specific solutions and not the architecture itself.

8.1. Constrained Devices

- C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover:
  - unable to manage complex authorization policies
  - unable to manage a large number of secure connections
  - without user interface
  - without constant network connectivity
  - unable to precisely measure time
  - required to save on wireless communication due to high power consumption
- CAS and AS are not assumed to be constrained devices.
- All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.
- Public key cryptography requires additional resources (such as RAM, ROM, power, specialized hardware).
- A solution will need to consider support for a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. Section 9.2).

8.2. Server-side Authorization

- RS enforces authorization for access to a resource based on credentials presented by C, the requested resource, the REST method, and local context in RS at the time of the request, or on any subset of this information.
- The authorization decision is enforced by RS.

    * RS needs to have authorization information in order to verify that C is allowed to access the resource as requested.

    * RS needs to make sure that it provides resource access only to authorized clients.

  - Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (for instance, resource descriptions).

8.3. Client-side Authorization Information

  - C enforces client-side authorization by protecting its requests to RS and by authenticating results from RS, making use of decisions and policies as well as keying material provided by CAS.

8.4. Resource Access

  - Resources are accessed in a RESTful manner using methods such as GET, PUT, POST, DELETE.

  - By default, the resource request needs to be integrity protected and may be encrypted end-to-end from C to RS. It needs to be possible for RS to detect a replayed request.

  - By default, the response to a request needs to be integrity protected and may be encrypted end-to-end from RS to C. It needs to be possible for C to detect a replayed response.

  - RS needs to be able to verify that the request comes from an authorized client.

  - C needs to be able to verify that the response to a request comes from the intended RS.

  - There may be resources whose access need not be protected (e.g. for discovery of the responsible AS).

8.5. Keys and Cipher Suites

  - A constrained node and its authorization manager (i.e., RS and AS, and C and CAS) have established cryptographic keys. For example, they share a secret key or each have the other's public key.

  - The transfer of authorization information is protected with symmetric and/or asymmetric keys.

- The access request/response is protected with symmetric and/or asymmetric keys.
- There must be a mechanism for RS to establish the necessary key(s) to verify and decrypt the request and to protect the response.
- There must be a mechanism for C to establish the necessary key(s) to protect the request and to verify and decrypt the response.
- There must be a mechanism for C to obtain the supported cipher suites of a RS.

8.6. Network Considerations

- A solution will need to consider network overload due to avoidable communication of a constrained node with its authorization manager (C with CAS, RS with AS).
- A solution will need to consider network overload by compact authorization information representation.
- A solution may want to optimize the case where authorization information does not change often.
- A solution should combine the mechanisms for providing authentication and authorization information to the client and RS where possible.
- A solution may consider support for an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

9. Security Considerations

This document discusses authorization-related tasks for constrained environments and describes how these tasks can be mapped to actors in the architecture.

In this section we focus on specific security aspects related to authorization in constrained-node networks. Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications of specific constraints on the security mechanisms employed. A wider view of security in constrained-node networks is provided in [I-D.irtf-t2trg-iot-seccons].

9.1. Physical Attacks on Sensor and Actuator Networks

The focus of this work is on constrained-node networks consisting of connected constrained devices such as sensors and actuators. The main function of such devices is to interact with the physical world by gathering information or performing an action. We now discuss attacks performed with physical access to such devices.

The main threats to sensors and actuator networks are:

- Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.

- Denial-of-service making the sensor/actuator unable to perform its intended task correctly.

  A number of attacks can be made with physical access to a device including probing attacks, timing attacks, power attacks, etc. However, with physical access to a sensor or actuator device it is possible to directly perform attacks equivalent of eavesdropping, manipulating data or denial of service. These attacks are possible by having physical access to the device, since the assets are related to the physical world. Moreover, this kind of attacks are in many cases straightforward (requires no special competence or tools, low cost given physical access, etc). If an attacker has full physical access to a sensor or actuator device, then much of the security functionality elaborated in this draft may not be effective to protect the asset during the physical attack.

9.2. Clocks and Time Measurements

Measuring time and keeping wall-clock time with certain accuracy is important to achieve certain security properties, for example to determine whether keying material an access token, or some other assertion, is valid. The required level of accuracy may differ for different applications.

Dynamic authorization in itself requires the ability to handle expiry or revocation of authorization decisions or to distinguish new authorization decisions from old.

For certain categories of devices we can assume that there is an internal clock which is sufficiently accurate to handle the time measurement requirements. If RS continuously measures time and can connect directly to AS, this relationship can be used to update RS in terms of time, removing some uncertainty, as well as to directly provide revocation information, removing authorizations that are no longer desired.

If RS continuously measures time but can't connect to AS or another trusted source of time, time drift may have to be accepted and it may be harder to manage revocation. However, RS may still be able to handle short lived access rights within some margins, by measuring the time since arrival of authorization information or request.

Some categories of devices in scope may be unable to measure time with any accuracy (e.g. because of sleep cycles). This category of devices is not suitable for the use cases which require measuring validity of assertions and authorizations in terms of absolute time such as TLS certificates but require a mechanism that is specifically designed for them.

10. IANA Considerations

This document has no actions for IANA.

11. Informative References

[HUM14delegation]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", 11th IEEE International Conference on Sensing, Communication, and Networking (SECON'14), June 30 - July 3, 2014.

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-14 (work in progress), January 2016.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-05 (work in progress), July 2018.

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

[I-D.irtf-t2trg-iot-seccons]
Garcia-Morchon, O., Kumar, S., and M. Sethi, "State-of-the-Art and Challenges for the Internet of Things Security", draft-irtf-t2trg-iot-seccons-15 (work in progress), May 2018.

[REST] Fielding, R. and R. Taylor, "Principled design of the modern Web architecture", ACM Trans. Inter. Tech. Vol. 2(2), pp. 115-150, DOI 10.1145/514183.514185, May 2002.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

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

[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>.

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

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

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

[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>.

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

[RFC8246] McManus, P., "HTTP Immutable Responses", RFC 8246, DOI 10.17487/RFC8246, September 2017, <https://www.rfc-editor.org/info/rfc8246>.

Acknowledgements

The authors would like to thank Olaf Bergmann, Robert Cragie, Samuel Erdtman, Klaus Hartke, Sandeep Kumar, John Mattson, Corinna Schmitt, Mohit Sethi, Abhinav Somaraju, Hannes Tschofenig, Vlasios Tsiatsis and Erik Wahlstroem for contributing to the discussion, giving helpful input and commenting on previous forms of this draft. The authors would also like to specifically acknowledge input provided by Hummen and others [HUM14delegation]. Robin Wilton provided extensive editorial comments that were the basis for significant improvements of the text.

Authors' Addresses

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

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

Ludwig Seitz
RISE SICS
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig.seitz@ri.se

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 20, 2018

M. Jones
Microsoft
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
ARM Ltd.

March 19, 2018

CBOR Web Token (CWT)
draft-ietf-ace-cbor-web-token-15

Abstract

CBOR Web Token (CWT) is a compact means of representing claims to be transferred between two parties. The claims in a CWT are encoded in the Concise Binary Object Representation (CBOR) and CBOR Object Signing and Encryption (COSE) is used for added application layer security protection. A claim is a piece of information asserted about a subject and is represented as a name/value pair consisting of a claim name and a claim value. CWT is derived from JSON Web Token (JWT) but uses CBOR rather than JSON.

Table of Contents

1. Introduction

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

An alternative encoding of claims is defined in this document. Instead of using JSON, as provided by JWTs, this specification uses CBOR [RFC7049] and calls this new structure "CBOR Web Token (CWT)", which is a compact means of representing secured claims to be transferred between two parties. CWT is closely related to JWT. It references the JWT claims and both its name and pronunciation are derived from JWT. To protect the claims contained in CWTs, the CBOR Object Signing and Encryption (COSE) [RFC8152] specification is used.

The suggested pronunciation of CWT is the same as the English word "cot".

1.1. CBOR Related Terminology

In JSON, maps are called objects and only have one kind of map key: a string. CBOR uses strings, negative integers, and unsigned integers as map keys. The integers are used for compactness of encoding and easy comparison. The inclusion of strings allows for an additional range of short encoded values to be used.

2. Terminology

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

14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document reuses terminology from JWT [RFC7519] and COSE [RFC8152].

StringOrURI
: The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate
: The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead of a JSON number. The encoding is modified so that the leading tag 1 (epoch-based date/time) MUST be omitted.

Claim Name
: The human-readable name used to identify a claim.

Claim Key
: The CBOR map key used to identify a claim.

Claim Value
: The CBOR map value representing the value of the claim.

CWT Claims Set
: The CBOR map that contains the claims conveyed by the CWT.

3. Claims

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

To keep CWTs as small as possible, the Claim Keys are represented using integers or text strings. Section 4 summarizes all keys used to identify the claims defined in this document.

3.1. Registered Claims

None of the claims defined below are intended to be mandatory to use or implement. They rather provide a starting point for a set of useful, interoperable claims. Applications using CWTs should define which specific claims they use and when they are required or optional.

3.1.1. iss (Issuer) Claim

The "iss" (issuer) claim has the same meaning and processing rules as the "iss" claim defined in Section 4.1.1 of [RFC7519], except that the value is a StringOrURI, as defined in Section 2 of this specification. The Claim Key 1 is used to identify this claim.

3.1.2. sub (Subject) Claim

The "sub" (subject) claim has the same meaning and processing rules as the "sub" claim defined in Section 4.1.2 of [RFC7519], except that the value is a StringOrURI, as defined in Section 2 of this specification. The Claim Key 2 is used to identify this claim.

3.1.3. aud (Audience) Claim

The "aud" (audience) claim has the same meaning and processing rules as the "aud" claim defined in Section 4.1.3 of [RFC7519], except that the value of the audience claim is a StringOrURI when it is not an array or each of the audience array element values is a StringOrURI when the audience claim value is an array. (StringOrURI is defined in Section 2 of this specification.) The Claim Key 3 is used to identify this claim.

3.1.4. exp (Expiration Time) Claim

The "exp" (expiration time) claim has the same meaning and processing rules as the "exp" claim defined in Section 4.1.4 of [RFC7519], except that the value is a NumericDate, as defined in Section 2 of this specification. The Claim Key 4 is used to identify this claim.

3.1.5. nbf (Not Before) Claim

The "nbf" (not before) claim has the same meaning and processing rules as the "nbf" claim defined in Section 4.1.5 of [RFC7519], except that the value is a NumericDate, as defined in Section 2 of this specification. The Claim Key 5 is used to identify this claim.

3.1.6. iat (Issued At) Claim

The "iat" (issued at) claim has the same meaning and processing rules as the "iat" claim defined in Section 4.1.6 of [RFC7519], except that the value is a NumericDate, as defined in Section 2 of this specification. The Claim Key 6 is used to identify this claim.

3.1.7. cti (CWT ID) Claim

The "cti" (CWT ID) claim has the same meaning and processing rules as the "jti" claim defined in Section 4.1.7 of [RFC7519], except that the value is a byte string. The Claim Key 7 is used to identify this claim.

4. Summary of the claim names, keys, and value types

Name	Key	Value type
iss	1	text string
sub	2	text string
aud	3	text string
exp	4	integer or floating-point number
nbf	5	integer or floating-point number
iat	6	integer or floating-point number
cti	7	byte string

Table 1: Summary of the claim names, keys, and value types

5. CBOR Tags and Claim Values

The claim values defined in this specification MUST NOT be prefixed with any CBOR tag. For instance, while CBOR tag 1 (epoch-based date/time) could logically be prefixed to values of the "exp", "nbf", and "iat" claims, this is unnecessary, since the representation of the claim values is already specified by the claim definitions. Tagging claim values would only take up extra space without adding information. However, this does not prohibit future claim definitions from requiring the use of CBOR tags for those specific claims.

6. CWT CBOR Tag

How to determine that a CBOR data structure is a CWT is application-dependent. In some cases, this information is known from the application context, such as from the position of the CWT in a data structure at which the value must be a CWT. One method of indicating

that a CBOR object is a CWT is the use of the "application/cwt" content type by a transport protocol.

This section defines the CWT CBOR tag as another means for applications to declare that a CBOR data structure is a CWT. Its use is optional and is intended for use in cases in which this information would not otherwise be known.

If present, the CWT tag MUST prefix a tagged object using one of the COSE CBOR tags. In this example, the COSE_Mac0 tag is used. The actual COSE_Mac0 object has been excluded from this example.

```
/ CWT CBOR tag / 61(
  / COSE_Mac0 CBOR tag / 17(
    / COSE_Mac0 object /
  )
)
```

Figure 1: Example of a CWT tag usage

7. Creating and Validating CWTs

7.1. Creating a CWT

To create a CWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Create a CWT Claims Set containing the desired claims.

2. Let the Message be the binary representation of the CWT Claims Set.

3. Create a COSE Header containing the desired set of Header Parameters. The COSE Header MUST be valid per the [RFC8152] specification.

4. Depending upon whether the CWT is signed, MACed, or encrypted, there are three cases:

   * If the CWT is signed, create a COSE_Sign/COSE_Sign1 object using the Message as the COSE_Sign/COSE_Sign1 Payload; all steps specified in [RFC8152] for creating a COSE_Sign/ COSE_Sign1 object MUST be followed.

   * Else, if the CWT is MACed, create a COSE_Mac/COSE_Mac0 object using the Message as the COSE_Mac/COSE_Mac0 Payload; all steps

specified in [RFC8152] for creating a COSE_Mac/COSE_Mac0 object MUST be followed.

* Else, if the CWT is a COSE_Encrypt/COSE_Encrypt0 object, create a COSE_Encrypt/COSE_Encrypt0 using the Message as the plaintext for the COSE_Encrypt/COSE_Encrypt0 object; all steps specified in [RFC8152] for creating a COSE_Encrypt/ COSE_Encrypt0 object MUST be followed.

5. If a nested signing, MACing, or encryption operation will be performed, let the Message be the tagged COSE_Sign/COSE_Sign1, COSE_Mac/COSE_Mac0, or COSE_Encrypt/COSE_Encrypt0, and return to Step 3.

6. If needed by the application, prepend the COSE object with the appropriate COSE CBOR tag to indicate the type of the COSE object. If needed by the application, prepend the COSE object with the CWT CBOR tag to indicate that the COSE object is a CWT.

7.2. Validating a CWT

When validating a CWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If any of the listed steps fail, then the CWT MUST be rejected -- that is, treated by the application as invalid input.

1. Verify that the CWT is a valid CBOR object.

2. If the object begins with the CWT CBOR tag, remove it and verify that one of the COSE CBOR tags follows it.

3. If the object is tagged with one of the COSE CBOR tags, remove it and use it to determine the type of the CWT, COSE_Sign/ COSE_Sign1, COSE_Mac/COSE_Mac0, or COSE_Encrypt/COSE_Encrypt0. If the object does not have a COSE CBOR tag, the COSE message type is determined from the application context.

4. Verify that the resulting COSE Header includes only parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.

5. Depending upon whether the CWT is a signed, MACed, or encrypted, there are three cases:

   * If the CWT is a COSE_Sign/COSE_Sign1, follow the steps specified in [RFC8152] Section 4 (Signing Objects) for

validating a COSE_Sign/COSE_Sign1 object. Let the Message be the COSE_Sign/COSE_Sign1 payload.

* Else, if the CWT is a COSE_Mac/COSE_Mac0, follow the steps specified in [RFC8152] Section 6 (MAC Objects) for validating a COSE_Mac/COSE_Mac0 object. Let the Message be the COSE_Mac/COSE_Mac0 payload.

* Else, if the CWT is a COSE_Encrypt/COSE_Encrypt0 object, follow the steps specified in [RFC8152] Section 5 (Encryption Objects) for validating a COSE_Encrypt/COSE_Encrypt0 object. Let the Message be the resulting plaintext.

6. If the Message begins with a COSE CBOR tag, then the Message is a CWT that was the subject of nested signing, MACing, or encryption operations. In this case, return to Step 1, using the Message as the CWT.

7. Verify that the Message is a valid CBOR map; let the CWT Claims Set be this CBOR map.

8. Security Considerations

The security of the CWT relies upon on the protections offered by COSE. Unless the claims in a CWT are protected, an adversary can modify, add, or remove claims.

Since the claims conveyed in a CWT may be used to make authorization decisions, it is not only important to protect the CWT in transit but also to ensure that the recipient can authenticate the party that assembled the claims and created the CWT. Without trust of the recipient in the party that created the CWT, no sensible authorization decision can be made. Furthermore, the creator of the CWT needs to carefully evaluate each claim value prior to including it in the CWT so that the recipient can be assured of the validity of the information provided.

While syntactically the signing and encryption operations for Nested CWTs may be applied in any order, if both signing and encryption are necessary, normally producers should sign the message and then encrypt the result (thus encrypting the signature). This prevents attacks in which the signature is stripped, leaving just an encrypted message, as well as providing privacy for the signer. Furthermore, signatures over encrypted text are not considered valid in many jurisdictions.

9. IANA Considerations

9.1. CBOR Web Token (CWT) Claims Registry

This section establishes the IANA "CBOR Web Token (CWT) Claims" registry.

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

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

Criteria that should be applied by the Designated Experts includes determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration description is clear. Registrations for the limited set of values between -256 and 255 and strings of length 1 are to be restricted to claims with general applicability.

IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list.

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

Since a high degree of overlap is expected between the contents of the "CBOR Web Token (CWT) Claims" registry and the "JSON Web Token Claims" registry, overlap in the corresponding pools of Designated Experts would be useful to help ensure that an appropriate level of coordination between the registries is maintained.

9.1.1. Registration Template

Claim Name:
: The human-readable name requested (e.g., "iss").

Claim Description:
: Brief description of the claim (e.g., "Issuer").

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

Claim Key:
: CBOR map key for the claim. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 and strings of length 1 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values greater than 65535 and strings of length greater than 2 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.

Claim Value Type(s):
: CBOR types that can be used for the claim value.

Change Controller:
: For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
: Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

9.1.2. Initial Registry Contents

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

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

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

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

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

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

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

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

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

9.2. Media Type Registration

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

9.2.1. Registry Contents

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

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

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

9.3. CoAP Content-Formats Registration

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

9.3.1. Registry Contents

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

9.4. CBOR Tag registration

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

9.4.1. Registry Contents

- CBOR Tag: TBD (maybe 61 to use the same value as the Content-Format)
- Data Item: CBOR Web Token (CWT)
- Semantics: CBOR Web Token (CWT), as defined in [[this specification]]
- Description of Semantics: [[this specification]]
- Point of Contact: Michael B. Jones, mbj@microsoft.com

10. References

10.1. Normative References

[IANA.CBOR.Tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>.

[IANA.CoAP.Content-Formats]
IANA, "CoAP Content-Formats", <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

[IANA.MediaTypes]
IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

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

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

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

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

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

10.2. Informative References

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

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

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

Appendix A. Examples

This appendix includes a set of CWT examples that show how the CWT Claims Set can be protected. There are examples that are signed, MACed, encrypted, and that use nested signing and encryption. To make the examples easier to read, they are presented both as hex strings and in the extended CBOR diagnostic notation described in Section 6 of [RFC7049].

Where a byte string is to carry an embedded CBOR-encoded item, the diagnostic notation for this CBOR data item can be enclosed in '<<' and '>>' to notate the byte string resulting from encoding the data item, e.g., h'63666F6F' translates to <<"foo">>.

A.1. Example CWT Claims Set

The CWT Claims Set used for the different examples displays usage of all the defined claims. For signed and MACed examples, the CWT Claims Set is the CBOR encoding as a byte string.

```
a70175636f61703a2f2f61732e6578616d706c652e636f6d02656572696b7703
7818636f61703a2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0
051a5610d9f0061a5610d9f007420b71
```

Figure 2: Example CWT Claims Set as hex string

```
{
  / iss / 1: "coap://as.example.com",
  / sub / 2: "erikw",
  / aud / 3: "coap://light.example.com",
  / exp / 4: 1444064944,
  / nbf / 5: 1443944944,
  / iat / 6: 1443944944,
  / cti / 7: h'0b71'
}
```

Figure 3: Example CWT Claims Set in CBOR diagnostic notation

A.2. Example keys

This section contains the keys used to sign, MAC, and encrypt the messages in this appendix. Line breaks are for display purposes only.

A.2.1. 128-bit Symmetric Key

```
a42050231f4c4d4d3051fdc2ec0a3851d5b3830104024c53796d6d6574726963
313238030a
```

Figure 4: 128-bit symmetric COSE_Key as hex string

```
{
  / k /   -1: h'231f4c4d4d3051fdc2ec0a3851d5b383'
  / kty /  1: 4 / Symmetric /,
  / kid /  2: h'53796d6d6574726963313238' / 'Symmetric128' /,
  / alg /  3: 10 / AES-CCM-16-64-128 /
}
```

Figure 5: 128-bit symmetric COSE_Key in CBOR diagnostic notation

A.2.2. 256-bit Symmetric Key

```
a4205820403697de87af64611c1d32a05dab0fe1fcb715a86ab435f1ec99192d
795693880104024c53796d6d6574726963323536030a
```

Figure 6: 256-bit symmetric COSE_Key as hex string

```
{
  / k /   -1: h'403697de87af64611c1d32a05dab0fe1fcb715a86ab435f1
               ec99192d79569388'
  / kty /  1: 4 / Symmetric /,
  / kid /  4: h'53796d6d6574726963323536' / 'Symmetric256' /,
  / alg /  3: 4 / HMAC 256/64 /
}
```

Figure 7: 256-bit symmetric COSE_Key in CBOR diagnostic notation

A.2.3. ECDSA P-256 256-bit COSE Key

```
a72358206c1382765aec5358f117733d281c1c7bdc39884d04a45a1e6c67c858
bc206c1922582060f7f1a780d8a783bfb7a2dd6b2796e8128dbbcef9d3d168db
9529971a36e7b9215820143329cce7868e416927599cf65a34f3ce2ffda55a7e
ca69ed8919a394d42f0f2001010202524173796d6d6574726963454344534132
35360326
```

Figure 8: ECDSA 256-bit COSE Key as hex string

```
{
  / d /   -4: h'6c1382765aec5358f117733d281c1c7bdc39884d04a45a1e
              6c67c858bc206c19',
  / y /   -3: h'60f7f1a780d8a783bfb7a2dd6b2796e8128dbbcef9d3d168
              db9529971a36e7b9',
  / x /   -2: h'143329cce7868e416927599cf65a34f3ce2ffda55a7eca69
              ed8919a394d42f0f',
  / crv / -1: 1 / P-256 /,
  / kty /  1: 2 / EC2 /,
  / kid /  2: h'4173796d6d6574726963454344534132353650a701756
              23536' / 'AsymmetricECDSA256' /,
  / alg /  3: -7 / ECDSA 256 /
}
```

Figure 9: ECDSA 256-bit COSE Key in CBOR diagnostic notation

A.3. Example Signed CWT

This section shows a signed CWT with a single recipient and a full CWT Claims Set.

The signature is generated using the private key listed in Appendix A.2.3 and it can be validated using the public key from Appendix A.2.3. Line breaks are for display purposes only.

```
d28443a10126a104524173796d6d6574726963454344534132353650a701756
36f61703a2f2f61732e6578616d706c652e636f6d02656572696b77037818636f
61703a2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0051a5610d
9f0061a5610d9f007420b7158405427c1ff28d23fbad1f29c4c7c6a555e601d6f
a29f9179bc3d7438bacaca5acd08c8d4d4f96131680c429a01f85951ecee743a5
2b9b63632c57209120e1c9e30
```

Figure 10: Signed CWT as hex string

```
   18(
     [
       / protected / << {
         / alg / 1: -7 / ECDSA 256 /
       } >>,
       / unprotected / {
         / kid / 4: h'4173796d6d657472696345434453413
                      23536' / 'AsymmetricECDSA256' /
       },
       / payload / << {
         / iss / 1: "coap://as.example.com",
         / sub / 2: "erikw",
         / aud / 3: "coap://light.example.com",
         / exp / 4: 1444064944,
         / nbf / 5: 1443944944,
         / iat / 6: 1443944944,
         / cti / 7: h'0b71'
       } >>,
       / signature / h'5427c1ff28d23fbad1f29c4c7c6a555e601d6fa29f
                       9179bc3d7438bacaca5acd08c8d4d4f96131680c42
                       9a01f85951ecee743a52b9b63632c57209120e1c9e
                       30'
     ]
   )
```

Figure 11: Signed CWT in CBOR diagnostic notation

A.4. Example MACed CWT

This section shows a MACed CWT with a single recipient, a full CWT Claims Set, and a CWT tag.

The MAC is generated using the 256-bit symmetric key from Appendix A.2.2 with a 64-bit truncation. Line breaks are for display purposes only.

```
d83dd18443a10104a1044c53796d6d6574726963323536585 0a70175636f6170
3a2f2f61732e6578616d706c652e636f6d02656572696b77037818636f61703a
2f2f6c696768742e6578616d706c652e636f6d041a5612aeb0051a5610d9f006
1a5610d9f007420b7148093101ef6d789200
```

Figure 12: MACed CWT with CWT tag as hex string

```
61(
  17(
    [
      / protected / << {
        / alg / 1: 4 / HMAC-256-64 /
      } >>,
      / unprotected / {
        / kid / 4: h'53796d6d6574726963323536' / 'Symmetric256' /
      },
      / payload / << {
        / iss / 1: "coap://as.example.com",
        / sub / 2: "erikw",
        / aud / 3: "coap://light.example.com",
        / exp / 4: 1444064944,
        / nbf / 5: 1443944944,
        / iat / 6: 1443944944,
        / cti / 7: h'0b71'
      } >>,
      / tag / h'093101ef6d789200'
    ]
  )
)
```

Figure 13: MACed CWT with CWT tag in CBOR diagnostic notation

A.5. Example Encrypted CWT

This section shows an encrypted CWT with a single recipient and a full CWT Claims Set.

The encryption is done with AES-CCM mode using the 128-bit symmetric key from Appendix A.2.1 with a 64-bit tag and 13-byte nonce, i.e., COSE AES-CCM-16-64-128. Line breaks are for display purposes only.

```
d08343a1010aa2044c53796d6d6574726963313238054d99a0d7846e762c49ff
e8a63e0b5858b918a11fd81e438b7f973d9e2e119bcb22424ba0f38a80f27562
f400ee1d0d6c0fdb559c02421fd384fc2ebe22d7071378b0ea7428fff157444d
45f7e6afcda1aae5f6495830c58627087fc5b4974f319a8707a635dd643b
```

Figure 14: Encrypted CWT as hex string

```
16(
  [
    / protected / << {
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963313238' / 'Symmetric128' /,
      / iv /  5: h'99a0d7846e762c49ffe8a63e0b'
    },
    / ciphertext / h'b918a11fd81e438b7f973d9e2e119bcb22424ba0f38
                     a80f27562f400ee1d0d6c0fdb559c02421fd384fc2e
                     be22d7071378b0ea7428fff157444d45f7e6afcda1a
                     ae5f6495830c58627087fc5b4974f319a8707a635dd
                     643b'
  ]
)
```

Figure 15: Encrypted CWT in CBOR diagnostic notation

A.6. Example Nested CWT

This section shows a Nested CWT, signed and then encrypted, with a single recipient and a full CWT Claims Set.

The signature is generated using the private ECDSA key from Appendix A.2.3 and it can be validated using the public ECDSA parts from Appendix A.2.3. The encryption is done with AES-CCM mode using the 128-bit symmetric key from Appendix A.2.1 with a 64-bit tag and 13-byte nonce, i.e., COSE AES-CCM-16-64-128. The content type is set to CWT to indicate that there are multiple layers of COSE protection before finding the CWT Claims Set. The decrypted ciphertext will be a COSE_sign1 structure. In this example, it is the same one as in Appendix A.3, i.e., a Signed CWT Claims Set. Note that there is no limitation to the number of layers; this is an example with two layers. Line breaks are for display purposes only.

```
d08343a1010aa2044c53796d6d6574726963313238054d4a0694c0e69ee6b595
6655c7b258b7f6b0914f993de822cc47e5e57a188d7960b528a747446fe12f0e
7de05650dec74724366763f167a29c002dfd15b34d8993391cf49bc91127f545
dba8703d66f5b7f1ae91237503d371e6333df9708d78c4fb8a8386c8ff09dc49
af768b23179deab78d96490a66d5724fb33900c60799d9872fac6da3bdb89043
d67c2a05414ce331b5b8f1ed8ff7138f45905db2c4d5bc8045ab372bff142631
610a7e0f677b7e9b0bc73adefdcee16d9d5d284c616abeab5d8c291ce0
```

Figure 16: Signed and Encrypted CWT as hex string

```
16(
  [
    / protected / << {
      / alg / 1: 10 / AES-CCM-16-64-128 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963313238' / 'Symmetric128' /,
      / iv /  5: h'4a0694c0e69ee6b5956655c7b2'
    },
    / ciphertext / h'f6b0914f993de822cc47e5e57a188d7960b528a7474
                     46fe12f0e7de05650dec74724366763f167a29c002d
                     fd15b34d8993391cf49bc91127f545dba8703d66f5b
                     7f1ae91237503d371e6333df9708d78c4fb8a8386c8
                     ff09dc49af768b23179deab78d96490a66d5724fb33
                     900c60799d9872fac6da3bdb89043d67c2a05414ce3
                     31b5b8f1ed8ff7138f45905db2c4d5bc8045ab372bf
                     f142631610a7e0f677b7e9b0bc73adefdcee16d9d5d
                     284c616abeab5d8c291ce0'
  ]
)
```

Figure 17: Signed and Encrypted CWT in CBOR diagnostic notation

A.7. Example MACed CWT with a floating-point value

This section shows a MACed CWT with a single recipient and a simple CWT Claims Set. The CWT Claims Set with a floating-point 'iat' value.

The MAC is generated using the 256-bit symmetric key from Appendix A.2.2 with a 64-bit truncation. Line breaks are for display purposes only.

```
d18443a10104a1044c53796d6d65747269633235364ba106fb41d584367c2000
0048b8816f34c0542892
```

Figure 18: MACed CWT with a floating-point value as hex string

```
17(
  [
    / protected / << {
      / alg / 1: 4 / HMAC-256-64 /
    } >>,
    / unprotected / {
      / kid / 4: h'53796d6d6574726963323536' / 'Symmetric256' /,
    },
    / payload / << {
      / iat / 6: 1443944944.5
    } >>,
    / tag / h'b8816f34c0542892'
  ]
)
```

Figure 19: MACed CWT with a floating-point value in CBOR diagnostic notation

Appendix B. Acknowledgements

This specification is based on JSON Web Token (JWT) [RFC7519], the authors of which also include Nat Sakimura and John Bradley. It also incorporates suggestions made by many people, including Carsten Bormann, Alissa Cooper, Esko Dijk, Benjamin Kaduk, Warren Kumari, Carlos Martinez, Alexey Melnikov, Kathleen Moriarty, Eric Rescorla, Dan Romascanu, Adam Roach, Kyle Rose, Jim Schaad, Ludwig Seitz, and Goeran Selander.

[[RFC Editor: Is it possible to preserve the non-ASCII spellings of the names Erik Wahlstroem and Goeran Selander in the final specification?]]

Appendix C. Document History

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

-15

- Added section references when the terms "NumericDate" and "StringOrURI" are used, as suggested by Adam Roach.

-14

- Cleaned up the descriptions of the numeric ranges of claim keys being registered in the registration template for the "CBOR Web Token (CWT) Claims" registry, as suggested by Adam Roach.

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

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

- Added the text "IANA must only accept registry updates from the Designated Experts and should direct all requests for registration to the review mailing list" from RFC 7519, as suggested by Amanda Baber of IANA, which is also intended to address Alexey Melnikov's comment.

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

- Acknowledged additional reviewers.

-13

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

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

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

- Acknowledged additional reviewers.

-12

- Updated the RFC 5226 reference to RFC 8126.

- Made the IANA registration criteria consistent across sections.

- Stated that registrations for the limited set of values between -256 and 255 and strings of length 1 are to be restricted to claims with general applicability.

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

- Asked the RFC Editor whether it is possible to preserve the non-ASCII spellings of the names Erik Wahlstroem and Goeran Selander in the final specification.

-11

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

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

- Changed references of the form "Section 4.1.1 of JWT <xref target="RFC7519"/>" to "Section 4.1.1 of <xref target="RFC7519"/>" so that rfcmarkup will generate correct external section reference links.

- Updated Acknowledgements.

-10

- Clarified that the audience claim value can be a single audience value or an array of audience values, just as is the case for the JWT "aud" claim.

- Clarified the nested CWT description.

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

-09

- Added key ID values to the examples.

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

-08

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

-07

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

-06

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

-05

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

- Say that CWT is derived from JWT, rather than CWT is a profile of JWT.
- Used CBOR type names in descriptions, rather than major/minor type numbers.
- Clarified the NumericDate and StringOrURI descriptions.
- Changed to allow CWT claim names to use values of any legal CBOR map key type.
- Changed to use the CWT tag to identify nested CWTs instead of the CWT content type.
- Added an example using a floating-point date value.
- Acknowledged reviewers.

-04

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

-03

- Reworked the examples to include signed, MACed, encrypted, and nested CWTs.
- Defined the CWT CBOR tag and explained its usage.

-02

- Added IANA registration for the application/cwt media type.
- Clarified the nested CWT language.
- Corrected nits identified by Ludwig Seitz.

-01

- Added IANA registration for CWT Claims.
- Added IANA registration for the application/cwt CoAP content-format type.
- Added Samuel Erdtman as an editor.
- Changed Erik’s e-mail address.

-00

- Created the initial working group version based on draft-wahlstroem-ace-cbor-web-token-00.

Authors' Addresses

Michael B. Jones
Microsoft

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

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

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

Phone: +46702691499
Email: erdtman@spotify.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2021

L. Seitz
Combitech
G. Selander
Ericsson
E. Wahlstroem

S. Erdtman
Spotify AB
H. Tschofenig
Arm Ltd.

March 8, 2021

Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth) draft-ietf-ace-oauth-authz-38

Abstract

This specification defines a framework for authentication and authorization in Internet of Things (IoT) environments called ACE-OAuth. The framework is based on a set of building blocks including OAuth 2.0 and the Constrained Application Protocol (CoAP), thus transforming a well-known and widely used authorization solution into a form suitable for IoT devices. Existing specifications are used where possible, but extensions are added and profiles are defined to better serve the IoT use cases.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 9, 2021.

Table of Contents

1. Introduction

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

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

Appendix A gives an overview of the constraints considered in this design, and a more detailed treatment of constraints can be found in [RFC7228]. This design aims to accommodate different IoT deployments and thus a continuous range of device and network capabilities.

Taking energy consumption as an example: At one end there are energy-harvesting or battery powered devices which have a tight power budget, on the other end there are mains-powered devices, and all levels in between.

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

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

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

2. Terminology

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

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

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

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

Note that the term "endpoint" is used here following its OAuth definition, which is to denote resources such as token and introspection at the AS and authz-info at the RS (see Section 5.10.1 for a definition of the authz-info endpoint). The CoAP [RFC7252]

definition, which is "An entity participating in the CoAP protocol" is not used in this specification.

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

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

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

3. Overview

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

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

Another building block is the lightweight web transfer protocol CoAP [RFC7252], for those communication environments where HTTP is not appropriate. CoAP typically runs on top of UDP, which further reduces overhead and message exchanges. While this specification defines extensions for the use of OAuth over CoAP, other underlying protocols are not prohibited from being supported in the future, such as HTTP/2 [RFC7540], Message Queuing Telemetry Transport (MQTT) [MQTT5.0], Bluetooth Low Energy (BLE) [BLE] and QUIC [I-D.ietf-quic-transport]. Note that this document specifies protocol exchanges in terms of RESTful verbs such as GET and POST. Future profiles using protocols that do not support these verbs MUST specify how the corresponding protocol messages are transmitted instead.

A third building block is the Concise Binary Object Representation (CBOR) [RFC8949], for encodings where JSON [RFC8259] is not sufficiently compact. CBOR is a binary encoding designed for small code and message size, which may be used for encoding of self contained tokens, and also for encoding payloads transferred in protocol messages.

A fourth building block is CBOR Object Signing and Encryption (COSE) [RFC8152], which enables object-level layer security as an alternative or complement to transport layer security (DTLS [RFC6347] or TLS [RFC8446]). COSE is used to secure self-contained tokens such as proof-of-possession (PoP) tokens, which are an extension to the OAuth bearer tokens. The default token format is defined in CBOR web token (CWT) [RFC8392]. Application layer security for CoAP using COSE can be provided with OSCORE [RFC8613].

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

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

3.1. OAuth 2.0

The OAuth 2.0 authorization framework enables a client to obtain scoped access to a resource with the permission of a resource owner. Authorization information, or references to it, is passed between the nodes using access tokens. These access tokens are issued to clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

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

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

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

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

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

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

Proof of Possession Tokens:
A token may be bound to a cryptographic key, which is then used to bind the token to a request authorized by the token. Such tokens are called proof-of-possession tokens (or PoP tokens).

The proof-of-possession (PoP) security concept used here assumes that the AS acts as a trusted third party that binds keys to tokens. In the case of access tokens, these so called PoP keys are then used by the client to demonstrate the possession of the secret to the RS when accessing the resource. The RS, when receiving an access token, needs to verify that the key used by the client matches the one bound to the access token. When this

specification uses the term "access token" it is assumed to be a PoP access token token unless specifically stated otherwise.

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

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

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

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

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

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

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

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

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

3.2. CoAP

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

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

CoAP supports application-layer fragmentation of the CoAP payloads through blockwise transfers [RFC7959]. However, blockwise transfer

does not increase the size limits of CoAP options, therefore data encoded in options has to be kept small.

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

One application of COSE is OSCORE [RFC8613], which provides end-to-end confidentiality, integrity and replay protection, and a secure binding between CoAP request and response messages. In OSCORE, the CoAP messages are wrapped in COSE objects and sent using CoAP.

This framework RECOMMENDS the use of CoAP as replacement for HTTP for use in constrained environments. For communication security this framework does not make an explicit protocol recommendation, since the choice depends on the requirements of the specific application. DTLS [RFC6347], [I-D.ietf-tls-dtls13] and OSCORE [RFC8613] are mentioned as examples, other protocols fulfilling the requirements from Section 6.5 are also applicable.

4. Protocol Interactions

The ACE framework is based on the OAuth 2.0 protocol interactions using the token endpoint and optionally the introspection endpoint. A client obtains an access token, and optionally a refresh token, from an AS using the token endpoint and subsequently presents the access token to an RS to gain access to a protected resource. In most deployments the RS can process the access token locally, however in some cases the RS may present it to the AS via the introspection endpoint to get fresh information. These interactions are shown in Figure 1. An overview of various OAuth concepts is provided in Section 3.1.

The OAuth 2.0 framework defines a number of "protocol flows" via grant types, which have been extended further with extensions to OAuth 2.0 (such as [RFC7521] and [RFC8628]). What grant types works best depends on the usage scenario and [RFC7744] describes many different IoT use cases but there are two preferred grant types, namely the Authorization Code Grant (described in Section 4.1 of [RFC7521]) and the Client Credentials Grant (described in Section 4.4 of [RFC7521]). The Authorization Code Grant is a good fit for use with apps running on smart phones and tablets that request access to IoT devices, a common scenario in the smart home environment, where

users need to go through an authentication and authorization phase (at least during the initial setup phase). The native apps guidelines described in [RFC8252] are applicable to this use case. The Client Credential Grant is a good fit for use with IoT devices where the OAuth client itself is constrained. In such a case, the resource owner has pre-arranged access rights for the client with the authorization server, which is often accomplished using a commissioning tool.

The consent of the resource owner, for giving a client access to a protected resource, can be provided dynamically as in the traditional OAuth flows, or it could be pre-configured by the resource owner as authorization policies at the AS, which the AS evaluates when a token request arrives. The resource owner and the requesting party (i.e., client owner) are not shown in Figure 1.

This framework supports a wide variety of communication security mechanisms between the ACE entities, such as client, AS, and RS. It is assumed that the client has been registered (also called enrolled or onboarded) to an AS using a mechanism defined outside the scope of this document. In practice, various techniques for onboarding have been used, such as factory-based provisioning or the use of commissioning tools. Regardless of the onboarding technique, this provisioning procedure implies that the client and the AS exchange credentials and configuration parameters. These credentials are used to mutually authenticate each other and to protect messages exchanged between the client and the AS.

It is also assumed that the RS has been registered with the AS, potentially in a similar way as the client has been registered with the AS. Established keying material between the AS and the RS allows the AS to apply cryptographic protection to the access token to ensure that its content cannot be modified, and if needed, that the content is confidentiality protected.

The keying material necessary for establishing communication security between C and RS is dynamically established as part of the protocol described in this document.

At the start of the protocol, there is an optional discovery step where the client discovers the resource server and the resources this server hosts. In this step, the client might also determine what permissions are needed to access the protected resource. A generic procedure is described in Section 5.1; profiles MAY define other procedures for discovery.

In Bluetooth Low Energy, for example, advertisements are broadcasted by a peripheral, including information about the primary services.

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

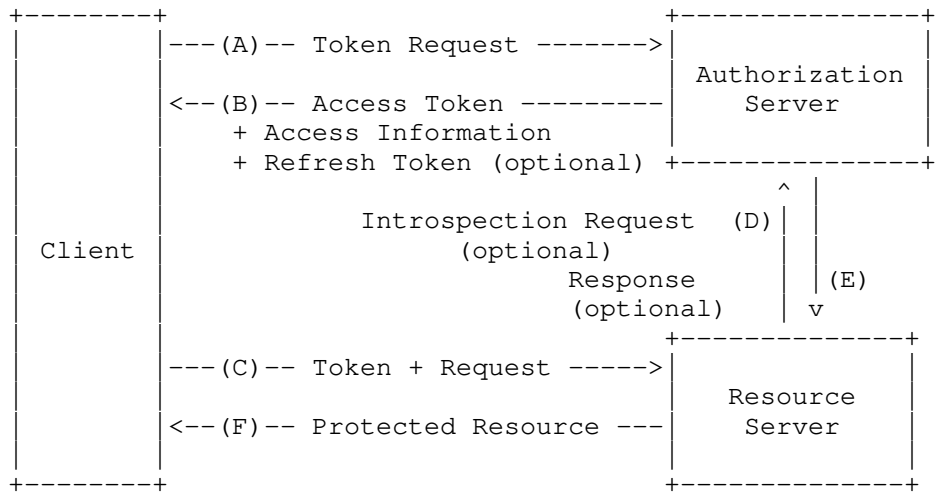


Figure 1: Basic Protocol Flow.

Requesting an Access Token (A):
The client makes an access token request to the token endpoint at the AS. This framework assumes the use of PoP access tokens (see Section 3.1 for a short description) wherein the AS binds a key to an access token. The client may include permissions it seeks to obtain, and information about the credentials it wants to use (e.g., symmetric/asymmetric cryptography or a reference to a specific credential).

Access Token Response (B):
If the AS successfully processes the request from the client, it returns an access token and optionally a refresh token (note that only certain grant types support refresh tokens). It can also return additional parameters, referred to as "Access Information". In addition to the response parameters defined by OAuth 2.0 and the PoP access token extension, this framework defines parameters that can be used to inform the client about capabilities of the RS, e.g. the profiles the RS supports. More information about these parameters can be found in Section 5.8.4.

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

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

(1) the client sends the access token containing, or referencing, the authorization information to the RS, that may be used for subsequent resource requests by the client, and

(2) the client makes the resource access request, using the communication security protocol and other Access Information obtained from the AS.

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

Token Introspection Request (D):
A resource server may be configured to introspect the access token by including it in a request to the introspection endpoint at that AS. Token introspection over CoAP is defined in Section 5.9 and for HTTP in [RFC7662].

Note that token introspection is an optional step and can be omitted if the token is self-contained and the resource server is prepared to perform the token validation on its own.

Token Introspection Response (E):
The AS validates the token and returns the most recent parameters, such as scope, audience, validity etc. associated with it back to the RS. The RS then uses the received parameters to process the request to either accept or to deny it.

Protected Resource (F):
: If the request from the client is authorized, the RS fulfills the request and returns a response with the appropriate response code. The RS uses the dynamically established keys to protect the response, according to the communication security protocol used.

5. Framework

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

Credential Provisioning
: For IoT, it cannot be assumed that the client and RS are part of a common key infrastructure, so the AS provisions credentials or associated information to allow mutual authentication between client and RS. The resulting security association between client and RS may then also be used to bind these credentials to the access tokens the client uses.

Proof-of-Possession
: The ACE framework, by default, implements proof-of-possession for access tokens, i.e., that the token holder can prove being a holder of the key bound to the token. The binding is provided by the "cnf" claim [RFC8747] indicating what key is used for proof-of-possession. If a client needs to submit a new access token, e.g., to obtain additional access rights, they can request that the AS binds this token to the same key as the previous one.

ACE Profiles
: The client or RS may be limited in the encodings or protocols it supports. To support a variety of different deployment settings, specific interactions between client and RS are defined in an ACE profile. In ACE framework the AS is expected to manage the matching of compatible profile choices between a client and an RS. The AS informs the client of the selected profile using the "ace_profile" parameter in the token response.

OAuth 2.0 requires the use of TLS both to protect the communication between AS and client when requesting an access token; between client and RS when accessing a resource and between AS and RS if introspection is used. In constrained settings TLS is not always feasible, or desirable. Nevertheless it is REQUIRED that the communications named above are encrypted, integrity protected and

protected against message replay. It is also REQUIRED that the communicating endpoints perform mutual authentication. Furthermore it MUST be assured that responses are bound to the requests in the sense that the receiver of a response can be certain that the response actually belongs to a certain request. Note that setting up such a secure communication may require some unprotected messages to be exchanged first (e.g. sending the token from the client to the RS).

Profiles MUST specify a communication security protocol between client and RS that provides the features required above. Profiles MUST specify a communication security protocol RECOMMENDED to be used between client and AS that provides the features required above. Profiles MUST specify for introspection a communication security protocol RECOMMENDED to be used between RS and AS that provides the features required above. These recommendations enable interoperability between different implementations without the need to define a new profile if the communication between C and AS, or between RS and AS, is protected with a different security protocol complying with the security requirements above.

In OAuth 2.0 the communication with the Token and the Introspection endpoints at the AS is assumed to be via HTTP and may use Uri-query parameters. When profiles of this framework use CoAP instead, it is REQUIRED to use of the following alternative instead of Uri-query parameters: The sender (client or RS) encodes the parameters of its request as a CBOR map and submits that map as the payload of the POST request.

Profiles that use CBOR encoding of protocol message parameters at the outermost encoding layer MUST use the media format 'application/ace+cbor'. If CoAP is used for communication, the Content-Format MUST be abbreviated with the ID: 19 (see Section 8.16).

The OAuth 2.0 AS uses a JSON structure in the payload of its responses both to client and RS. If CoAP is used, it is REQUIRED to use CBOR [RFC8949] instead of JSON. Depending on the profile, the CBOR payload MAY be enclosed in a non-CBOR cryptographic wrapper.

5.1. Discovering Authorization Servers

C must discover the AS in charge of RS to determine where to request the access token. To do so, C must 1. find out the AS URI to which the token request message must be sent and 2. MUST validate that the AS with this URI is authorized to provide access tokens for this RS.

In order to determine the AS URI, C MAY send an initial Unauthorized Resource Request message to RS. RS then denies the request and sends

the address of its AS back to C (see Section 5.2). How C validates the AS authorization is not in scope for this document. C may, e.g., ask it's owner if this AS is authorized for this RS. C may also use a mechanism that addresses both problems at once.

5.2. Unauthorized Resource Request Message

An Unauthorized Resource Request message is a request for any resource hosted by RS for which the client does not have authorization granted. RSes MUST treat any request for a protected resource as an Unauthorized Resource Request message when any of the following hold:

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

Note: These conditions ensure that the RS can handle requests autonomously once access was granted and a secure channel has been established between C and RS. The authz-info endpoint, as part of the process for authorizing to protected resources, is not itself a protected resource and MUST NOT be protected as specified above (cf. Section 5.10.1).

Unauthorized Resource Request messages MUST be denied with an "unauthorized_client" error response. In this response, the Resource Server SHOULD provide proper AS Request Creation Hints to enable the Client to request an access token from RS's AS as described in Section 5.3.

The handling of all client requests (including unauthorized ones) by the RS is described in Section 5.10.2.

5.3. AS Request Creation Hints

The AS Request Creation Hints message is sent by an RS as a response to an Unauthorized Resource Request message (see Section 5.2) to help the sender of the Unauthorized Resource Request message acquire a valid access token. The AS Request Creation Hints message is a CBOR map, with an OPTIONAL element "AS" specifying an absolute URI (see Section 4.3 of [RFC3986]) that identifies the appropriate AS for the RS.

The message can also contain the following OPTIONAL parameters:

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

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

- A "cnonce" element containing a client-nonce. See Section 5.3.1.

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

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

Name	CBOR Key	Value Type
AS	1	text string
kid	2	byte string
audience	5	text string
scope	9	text or byte string
cnonce	39	byte string

Figure 2: AS Request Creation Hints

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

Figure 3 shows an example for an AS Request Creation Hints message payload using CBOR [RFC8949] diagnostic notation, using the parameter names instead of the CBOR keys for better human readability.

```
4.01 Unauthorized
Content-Format: application/ace+cbor
Payload :
{
 "AS" : "coaps://as.example.com/token",
 "audience" : "coaps://rs.example.com"
 "scope" : "rTempC",
 "cnonce" : h'e0a156bb3f'
}
```

Figure 3: AS Request Creation Hints payload example

In the example above, the response parameter "AS" points the receiver of this message to the URI "coaps://as.example.com/token" to request access tokens. The RS sending this response (i.e., RS) uses an internal clock that is only loosely synchronized with the clock of the AS. Therefore it can not reliably verify the expiration time of access tokens it receives. To ensure a certain level of access token freshness nevetheless, the RS has included a "cnonce" parameter (see Section 5.3.1) in the response.

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

```
a4                                  # map(4)
   01                               # unsigned(1) (=AS)
   78 1c                            # text(28)
      636f6170733a2f2f61732e657861
      6d706c652e636f6d2f746f6b656e  # "coaps://as.example.com/token"
   05                               # unsigned(5) (=audience)
   76                               # text(22)
      636f6170733a2f2f72732e657861
      6d706c652e636f6d              # "coaps://rs.example.com"
   09                               # unsigned(9) (=scope)
   66                               # text(6)
      7254656d7043                  # "rTempC"
   18 27                            # unsigned(39) (=cnonce)
   45                               # bytes(5)
      e0a156bb3f                    #
```

Figure 4: AS Request Creation Hints example encoded in CBOR

5.3.1. The Client-Nonce Parameter

If the RS does not synchronize its clock with the AS, it could be tricked into accepting old access tokens, that are either expired or have been compromised. In order to ensure some level of token freshness in that case, the RS can use the "cnonce" (client-nonce)

parameter. The processing requirements for this parameter are as follows:

- An RS sending a "cnonce" parameter in an AS Request Creation Hints message MUST store information to validate that a given cnonce is fresh. How this is implemented internally is out of scope for this specification. Expiration of client-nonces should be based roughly on the time it would take a client to obtain an access token after receiving the AS Request Creation Hints message, with some allowance for unexpected delays.

- A client receiving a "cnonce" parameter in an AS Request Creation Hints message MUST include this in the parameters when requesting an access token at the AS, using the "cnonce" parameter from Section 5.8.4.4.

- If an AS grants an access token request containing a "cnonce" parameter, it MUST include this value in the access token, using the "cnonce" claim specified in Section 5.10.

- An RS that is using the client-nonce mechanism and that receives an access token MUST verify that this token contains a cnonce claim, with a client-nonce value that is fresh according to the information stored at the first step above. If the cnonce claim is not present or if the cnonce claim value is not fresh, the RS MUST discard the access token. If this was an interaction with the authz-info endpoint the RS MUST also respond with an error message using a response code equivalent to the CoAP code 4.01 (Unauthorized).

5.4. Authorization Grants

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

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

The grant type is selected depending on the use case. In cases where the client acts on behalf of the resource owner, the authorization code grant is recommended. If the client acts on behalf of the resource owner, but does not have any display or has very limited interaction possibilities, it is recommended to use the device code

grant defined in [RFC8628]. In cases where the client acts autonomously the client credentials grant is recommended.

For details on the different grant types, see section 1.3 of [RFC6749]. The OAuth 2.0 framework provides an extension mechanism for defining additional grant types, so profiles of this framework MAY define additional grant types, if needed.

5.5. Client Credentials

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

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

The OAuth framework defines one client credential type in section 2.3.1 of [RFC6749]: client id and client secret. [I-D.erdtman-ace-rpcc] adds raw-public-key and pre-shared-key to the client credentials types. Profiles of this framework MAY extend with an additional client credentials type using client certificates.

5.6. AS Authentication

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

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

5.7. The Authorization Endpoint

The OAuth 2.0 authorization endpoint is used to interact with the resource owner and obtain an authorization grant, in certain grant flows. The primary use case for the ACE-OAuth framework is for machine-to-machine interactions that do not involve the resource owner in the authorization flow; therefore, this endpoint is out of scope here. Future profiles may define constrained adaptation mechanisms for this endpoint as well. Non-constrained clients interacting with constrained resource servers can use the specification in section 3.1 of [RFC6749] and the attack countermeasures suggested in section 4.2 of [RFC6819].

5.8. The Token Endpoint

In standard OAuth 2.0, the AS provides the token endpoint for submitting access token requests. This framework extends the functionality of the token endpoint, giving the AS the possibility to help the client and RS to establish shared keys or to exchange their public keys. Furthermore, this framework defines encodings using CBOR, as a substitute for JSON.

The endpoint may, however, be exposed over HTTPS as in classical OAuth or even other transports. A profile MUST define the details of the mapping between the fields described below, and these transports. If HTTPS is used, JSON or CBOR payloads may be supported. If JSON payloads are used, the semantics of Section 4 of the OAuth 2.0 specification MUST be followed (with additions as described below). If CBOR payload is supported, the semantics described below MUST be followed.

For the AS to be able to issue a token, the client MUST be authenticated and present a valid grant for the scopes requested. Profiles of this framework MUST specify how the AS authenticates the client and how the communication between client and AS is protected, fulfilling the requirements specified in Section 5.

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

The figures of this section use CBOR diagnostic notation without the integer abbreviations for the parameters or their values for illustrative purposes. Note that implementations MUST use the integer abbreviations and the binary CBOR encoding, if the CBOR encoding is used.

5.8.1. Client-to-AS Request

The client sends a POST request to the token endpoint at the AS. The profile MUST specify how the communication is protected. The content of the request consists of the parameters specified in the relevant subsection of section 4 of the OAuth 2.0 specification [RFC6749], depending on the grant type, with the following exceptions and additions:

- The parameter "grant_type" is OPTIONAL in the context of this framework (as opposed to REQUIRED in RFC6749). If that parameter is missing, the default value "client_credentials" is implied.

- The "audience" parameter from [RFC8693] is OPTIONAL to request an access token bound to a specific audience.

- The "cnonce" parameter defined in Section 5.8.4.4 is REQUIRED if the RS provided a client-nonce in the "AS Request Creation Hints" message Section 5.3

- The "scope" parameter MAY be encoded as a byte string instead of the string encoding specified in section 3.3 of [RFC6749], in order allow compact encoding of complex scopes. The syntax of such a binary encoding is explicitly not specified here and left to profiles or applications, specifically note that a binary encoded scope does not necessarily use the space character '0x20' to delimit scope-tokens.

- The client can send an empty (null value) "ace_profile" parameter to indicate that it wants the AS to include the "ace_profile" parameter in the response. See Section 5.8.4.3.

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

The default behavior, is that the AS generates a symmetric proof-of-possession key for the client. In order to use an asymmetric key pair or to re-use a key previously established with the RS, the client is supposed to use the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

If CBOR is used then these parameters MUST be provided as a CBOR map.

When HTTP is used as a transport then the client makes a request to the token endpoint by sending the parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body, as defined in section 3.2 of [RFC6749].

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

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

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

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

Figure 6 shows a request for a token with an asymmetric proof-of-possession key. Note that in this example OSCORE [RFC8613] is used to provide object-security, therefore the Content-Format is "application/oscore" wrapping the "application/ace+cbor" type content. The OSCORE option has a decoded interpretation appended in parentheses for the reader's convenience. Also note that in this example the audience is implicitly known by both client and AS. Furthermore note that this example uses the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
OSCORE: 0x09, 0x05, 0x44, 0x6C
  (h=0, k=1, n=001, partialIV= 0x05, kid=[0x44, 0x6C])
Content-Format: "application/oscore"
Payload:
  0x44025d1 ... (full payload omitted for brevity) ... 68b3825e

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

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

Figure 7 shows a request for a token where a previously communicated proof-of-possession key is only referenced using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params].

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

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

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

5.8.2. AS-to-Client Response

If the access token request has been successfully verified by the AS and the client is authorized to obtain an access token corresponding to its access token request, the AS sends a response with the response code equivalent to the CoAP response code 2.01 (Created). If client request was invalid, or not authorized, the AS returns an error response as described in Section 5.8.3.

Note that the AS decides which token type and profile to use when issuing a successful response. It is assumed that the AS has prior knowledge of the capabilities of the client and the RS (see Appendix D). This prior knowledge may, for example, be set by the use of a dynamic client registration protocol exchange [RFC7591]. If the client has requested a specific proof-of-possession key using the "req_cnf" parameter from [I-D.ietf-ace-oauth-params], this may also influence which profile the AS selects, as it needs to support the use of the key type requested the client.

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

ace_profile:
: OPTIONAL unless the request included an empty ace_profile parameter in which case it is MANDATORY. This indicates the profile that the client MUST use towards the RS. See Section 5.8.4.3 for the formatting of this parameter. If this parameter is absent, the AS assumes that the client implicitly knows which profile to use towards the RS.

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

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

Figure 8 summarizes the parameters that can currently be part of the Access Information. Future extensions may define additional parameters.

Parameter name	Specified in
access_token	RFC 6749
token_type	RFC 6749
expires_in	RFC 6749
refresh_token	RFC 6749
scope	RFC 6749
state	RFC 6749
error	RFC 6749
error_description	RFC 6749
error_uri	RFC 6749
ace_profile	[this document]
cnf	[I-D.ietf-ace-oauth-params]
rs_cnf	[I-D.ietf-ace-oauth-params]

Figure 8: Access Information parameters

Figure 9 shows a response containing a token and a "cnf" parameter with a symmetric proof-of-possession key, which is defined in [I-D.ietf-ace-oauth-params]. Note that the key identifier 'kid' is only used to simplify indexing and retrieving the key, and no assumptions should be made that it is unique in the domains of either the client or the RS.

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

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

5.8.3. Error Response

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

- When using CBOR the raw payload before being processed by the communication security protocol MUST be encoded as a CBOR map.
- A response code equivalent to the CoAP code 4.00 (Bad Request) MUST be used for all error responses, except for invalid_client where a response code equivalent to the CoAP code 4.01 (Unauthorized) MAY be used under the same conditions as specified in Section 5.2 of [RFC6749].
- The Content-Format (for CoAP-based interactions) or media type (for HTTP-based interactions) "application/ace+cbor" MUST be used for the error response.

- The parameters "error", "error_description" and "error_uri" MUST be abbreviated using the codes specified in Figure 12, when a CBOR encoding is used.

- The error code (i.e., value of the "error" parameter) MUST be abbreviated as specified in Figure 10, when a CBOR encoding is used.

Name	CBOR Values
invalid_request	1
invalid_client	2
invalid_grant	3
unauthorized_client	4
unsupported_grant_type	5
invalid_scope	6
unsupported_pop_key	7
incompatible_ace_profiles	8

Figure 10: CBOR abbreviations for common error codes

In addition to the error responses defined in OAuth 2.0, the following behavior MUST be implemented by the AS:

- If the client submits an asymmetric key in the token request that the RS cannot process, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "unsupported_pop_key" defined in Figure 10.

- If the client and the RS it has requested an access token for do not share a common profile, the AS MUST reject that request with a response code equivalent to the CoAP code 4.00 (Bad Request) including the error code "incompatible_ace_profiles" defined in Figure 10.

5.8.4. Request and Response Parameters

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

5.8.4.1. Grant Type

The abbreviations specified in the registry defined in Section 8.5 MUST be used in CBOR encodings instead of the string values defined in [RFC6749], if CBOR payloads are used.

Name	CBOR Value	Original Specification
password	0	[RFC6749]
authorization_code	1	[RFC6749]
client_credentials	2	[RFC6749]
refresh_token	3	[RFC6749]

Figure 11: CBOR abbreviations for common grant types

5.8.4.2. Token Type

The "token_type" parameter, defined in section 5.1 of [RFC6749], allows the AS to indicate to the client which type of access token it is receiving (e.g., a bearer token).

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

The values in the "token_type" parameter MUST use the CBOR abbreviations defined in the registry specified by Section 8.7, if a CBOR encoding is used.

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

5.8.4.3. Profile

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

A profile MUST specify an identifier that MUST be used to uniquely identify itself in the "ace_profile" parameter. The textual representation of the profile identifier is intended for human readability and for JSON-based interactions, it MUST NOT be used for

CBOR-based interactions. Profiles MUST register their identifier in the registry defined in Section 8.8.

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

Clients that want the AS to provide them with the "ace_profile" parameter in the access token response can indicate that by sending a ace_profile parameter with a null value (for CBOR-based interactions) or an empty string (for JSON based interactions) in the access token request.

5.8.4.4. Client-Nonce

This parameter MUST be sent from the client to the AS, if it previously received a "cnonce" parameter in the AS Request Creation Hints Section 5.3. The parameter is encoded as a byte string for CBOR-based interactions, and as a string (Base64 encoded binary) for JSON-based interactions. It MUST copy the value from the cnonce parameter in the AS Request Creation Hints.

5.8.5. Mapping Parameters to CBOR

If CBOR encoding is used, all OAuth parameters in access token requests and responses MUST be mapped to CBOR types as specified in the registry defined by Section 8.10, using the given integer abbreviation for the map keys.

Note that we have aligned the abbreviations corresponding to claims with the abbreviations defined in [RFC8392].

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

Name	CBOR Key	Value Type
access_token	1	byte string
expires_in	2	unsigned integer
audience	5	text string
scope	9	text or byte string
client_id	24	text string
client_secret	25	byte string
response_type	26	text string
redirect_uri	27	text string
state	28	text string
code	29	byte string
error	30	integer
error_description	31	text string
error_uri	32	text string
grant_type	33	unsigned integer
token_type	34	integer
username	35	text string
password	36	text string
refresh_token	37	byte string
ace_profile	38	integer
cnonce	39	byte string

Figure 12: CBOR mappings used in token requests and responses

5.9. The Introspection Endpoint

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

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

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

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

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

5.9.1. Introspection Request

The requesting entity sends a POST request to the introspection endpoint at the AS. The profile MUST specify how the communication is protected. If CBOR is used, the payload MUST be encoded as a CBOR map with a "token" entry containing the access token. Further optional parameters representing additional context that is known by the requesting entity to aid the AS in its response MAY be included.

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

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

For example, Figure 13 shows an RS calling the token introspection endpoint at the AS to query about an OAuth 2.0 proof-of-possession token. Note that object security based on OSCORE [RFC8613] is assumed in this example, therefore the Content-Format is "application/oscore". Figure 14 shows the decoded payload.

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

Figure 13: Example introspection request.

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

Figure 14: Decoded payload.

5.9.2. Introspection Response

If the introspection request is authorized and successfully processed, the AS sends a response with the response code equivalent to the CoAP code 2.01 (Created). If the introspection request was invalid, not authorized or couldn't be processed the AS returns an error response as described in Section 5.9.3.

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

ace_profile OPTIONAL. This indicates the profile that the RS MUST use with the client. See Section 5.8.4.3 for more details on the formatting of this parameter.

cnonce OPTIONAL. A client-nonce provided to the AS by the client. The RS MUST verify that this corresponds to the client-nonce previously provided to the client in the AS Request Creation Hints. See Section 5.3 and Section 5.8.4.4.

exi OPTIONAL. The "expires-in" claim associated to this access token. See Section 5.10.3.

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

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

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

Figure 15: Example introspection response.

5.9.3. Error Response

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

- If content is sent and CBOR is used the payload MUST be encoded as a CBOR map and the Content-Format "application/ace+cbor" MUST be used.

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

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

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

- The error codes MUST be abbreviated using the codes specified in the registry defined by Section 8.4.

Note that a properly formed and authorized query for an inactive or otherwise invalid token does not warrant an error response by this specification. In these cases, the authorization server MUST instead

respond with an introspection response with the "active" field set to "false".

5.9.4. Mapping Introspection parameters to CBOR

If CBOR is used, the introspection request and response parameters MUST be mapped to CBOR types as specified in the registry defined by Section 8.12, using the given integer abbreviation for the map key.

Note that we have aligned abbreviations that correspond to a claim with the abbreviations defined in [RFC8392] and the abbreviations of parameters with the same name from Section 5.8.5.

Parameter name	CBOR Key	Value Type
iss	1	text string
sub	2	text string
aud	3	text string
exp	4	integer or floating-point number
nbf	5	integer or floating-point number
iat	6	integer or floating-point number
cti	7	byte string
scope	9	text or byte string
active	10	True or False
token	11	byte string
client_id	24	text string
error	30	integer
error_description	31	text string
error_uri	32	text string
token_type_hint	33	text string
token_type	34	integer
username	35	text string
ace_profile	38	integer
cnonce	39	byte string
exi	40	unsigned integer

Figure 16: CBOR Mappings to Token Introspection Parameters.

5.10. The Access Token

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

In order to facilitate offline processing of access tokens, this document uses the "cnf" claim from [RFC8747] and the "scope" claim from [RFC8693] for JWT- and CWT-encoded tokens. In addition to string encoding specified for the "scope" claim, a binary encoding MAY be used. The syntax of such an encoding is explicitly not specified here and left to profiles or applications, specifically note that a binary encoded scope does not necessarily use the space character '0x20' to delimit scope-tokens.

If the AS needs to convey a hint to the RS about which profile it should use to communicate with the client, the AS MAY include an "ace_profile" claim in the access token, with the same syntax and semantics as defined in Section 5.8.4.3.

If the client submitted a client-nonce parameter in the access token request Section 5.8.4.4, the AS MUST include the value of this parameter in the "cnonce" claim specified here. The "cnonce" claim uses binary encoding.

5.10.1. The Authorization Information Endpoint

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

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

The method consists of an authz-info endpoint, implemented by the RS. A client using this method MUST make a POST request to the authz-info endpoint at the RS with the access token in the payload. The RS receiving the token MUST verify the validity of the token. If the token is valid, the RS MUST respond to the POST request with 2.01 (Created). Section Section 5.10.1.1 outlines how an RS MUST proceed to verify the validity of an access token.

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

This specification RECOMMENDS that an RS stores only one token per proof-of-possession key. This means that an additional token linked to the same key will supersede any existing token at the RS, by replacing the corresponding authorization information. The reason is that this greatly simplifies (constrained) implementations, with

respect to required storage and resolving a request to the applicable token.

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

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

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

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

5.10.1.1. Verifying an Access Token

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

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

In case the token is communicated by reference the RS needs to obtain the claims first. When the RS uses token introspection the relevant specification is [RFC7662] with CoAP transport specified in Section 5.9.

Errors may happen during this initial processing stage:

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

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

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

iss The issuer claim must identify an AS that has the authority to issue access tokens for the receiving RS. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.01 (Unauthorized).

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

aud The audience claim must refer to an audience that the RS identifies with. If that is not the case the RS MUST discard the token. If this was an interaction with authz-info, the RS MUST also respond with a response code equivalent to the CoAP code 4.03 (Forbidden).

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

Additional processing may be needed for other claims in a way specific to a profile or the underlying application.

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

Also note that profiles of this framework may define access token transport mechanisms that do not allow for error responses. Therefore the error messages specified here only apply if the token was sent to the authz-info endpoint.

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

5.10.1.2. Protecting the Authorization Information Endpoint

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

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

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

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

5.10.2. Client Requests to the RS

Before sending a request to an RS, the client MUST verify that the keys used to protect this communication are still valid. See Section 5.10.4 for details on how the client determines the validity of the keys used.

If an RS receives a request from a client, and the target resource requires authorization, the RS MUST first verify that it has an access token that authorizes this request, and that the client has performed the proof-of-possession binding that token to the request.

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

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

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

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

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

5.10.3. Token Expiration

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

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

- The RS verifies the validity of the token by performing an introspection request as specified in Section 5.9. This requires the RS to have a reliable network connection to the AS and to be able to handle two secure sessions in parallel (C to RS and RS to AS).

- In order to support token expiration for devices that have no reliable way of synchronizing their internal clocks, this specification defines the following approach: The claim "exi" ("expires in") can be used, to provide the RS with the lifetime of the token in seconds from the time the RS first receives the token. For CBOR-based interaction this parameter is encoded as unsigned integer, while JSON-based interactions encode this as JSON number.

- Processing this claim requires that the RS does the following:

  * For each token the RS receives, that contains an "exi" claim: Keep track of the time it received that token and revisit that list regularly to expunge expired tokens.

  * Keep track of the identifiers of tokens containing the "exi" claim that have expired (in order to avoid accepting them again). In order to avoid an unbounded memory usage growth, this MUST be implemented in the following way when the "exi" claim is used:

    + When creating the token, the AS MUST add a 'cti' claim (or 'jti' for JWTs) to the access token. The value of this claim MUST be created as the binary representation of the concatenation of the identifier of the RS with a sequence number counting the tokens containing an 'exi' claim, issued by this AS for the RS.

    + The RS MUST store the highest sequence number of an expired token containing the "exi" claim that it has seen, and treat tokens with lower sequence numbers as expired.

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

5.10.4. Key Expiration

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

properly protected and could possibly have been forged by an attacker.

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

- The client knows a default validity time for all tokens it is using (i.e. how long a token is valid after being issued). This information could be provisioned to the client when it is registered at the AS, or published by the AS in a way that the client can query.

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

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

6. Security Considerations

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

The following subsections address issues specific to this document and it's use in constrained environments.

6.1. Protecting Tokens

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

If the token is intended for multiple recipients (i.e. an audience that is a group), integrity protection of the token with a symmetric

key, shared between the AS and the recipients, is not sufficient, since any of the recipients could modify the token undetected by the other recipients. Therefore a token with a multi-recipient audience MUST be protected with an asymmetric signature.

It is important for the authorization server to include the identity of the intended recipient (the audience), typically a single resource server (or a list of resource servers), in the token. The same shared secret MUST NOT be used as proof-of-possession key with multiple resource servers since the benefit from using the proof-of-possession concept is then significantly reduced.

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

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

In certain situations it may be necessary to revoke an access token that is still valid. Client-initiated revocation is specified in [RFC7009] for OAuth 2.0. Other revocation mechanisms are currently not specified, as the underlying assumption in OAuth is that access tokens are issued with a relatively short lifetime. This may not hold true for disconnected constrained devices, needing access tokens with relatively long lifetimes, and would therefore necessitate further standardization work that is out of scope for this document.

6.2. Communication Security

Communication with the authorization server MUST use confidentiality protection. This step is extremely important since the client or the RS may obtain the proof-of-possession key from the authorization server for use with a specific access token. Not using confidentiality protection exposes this secret (and the access token) to an eavesdropper thereby completely negating proof-of-possession security. The requirements for communication security of profiles are specified in Section 5.

Additional protection for the access token can be applied by encrypting it, for example encryption of CWTs is specified in Section 5.1 of [RFC8392]. Such additional protection can be necessary if the token is later transferred over an insecure connection (e.g. when it is sent to the authz-info endpoint).

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

6.3. Long-Term Credentials

Both clients and RSs have long-term credentials that are used to secure communications, and authenticate to the AS. These credentials need to be protected against unauthorized access. In constrained devices, deployed in publicly accessible places, such protection can be difficult to achieve without specialized hardware (e.g. secure key storage memory).

If credentials are lost or compromised, the operator of the affected devices needs to have procedures to invalidate any access these credentials give and to revoke tokens linked to such credentials. The loss of a credential linked to a specific device MUST NOT lead to a compromise of other credentials not linked to that device, therefore secret keys used for authentication MUST NOT be shared between more than two parties.

Operators of clients or RS SHOULD have procedures in place to replace credentials that are suspected to have been compromised or that have been lost.

Operators also SHOULD have procedures for decommissioning devices, that include securely erasing credentials and other security critical material in the devices being decommissioned.

6.4. Unprotected AS Request Creation Hints

Initially, no secure channel exists to protect the communication between C and RS. Thus, C cannot determine if the AS Request Creation Hints contained in an unprotected response from RS to an unauthorized request (see Section 5.3) are authentic. C therefore MUST determine if an AS is authorized to provide access tokens for a certain RS.

A compromised RS may use the hints for attempting to trick a client into contacting an AS that is not supposed to be in charge of that RS. Therefore, C must not communicate with an AS if it cannot determine that this AS has the authority to issue access tokens for this RS. Otherwise, a compromised RS may use this to perform a

denial of service attack against a specific AS, by redirecting a large number of client requests to that AS.

6.5. Minimal security requirements for communication

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

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

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

C-RS The initial communication between the client and the Resource Server can not be secured in general, since the RS is not in possession of on access token for that client, which would carry the necessary parameters. If both parties support DTLS without client authentication it is RECOMMEND to use this mechanism for protecting the initial communication. After the client has successfully transmitted the access token to the RS, a secure communication protocol MUST be established between client and RS for the actual resource request. This protocol MUST provide confidentiality, integrity and replay protection as well as a binding between requests and responses. This requires that the client learned either the RS's public key or received a symmetric proof-of-possession key bound to the access token from the AS. The RS must have learned either the client's public key or a shared symmetric key from the claims in the token or an

introspection request. Since ACE does not provide profile negotiation between C and RS, the client MUST have learned what profile the RS supports (e.g. from the AS or pre-configured) and initiate the communication accordingly.

6.6. Token Freshness and Expiration

An RS that is offline faces the problem of clock drift. Since it cannot synchronize its clock with the AS, it may be tricked into accepting old access tokens that are no longer valid or have been compromised. In order to prevent this, an RS may use the nonce-based mechanism defined in Section 5.3 to ensure freshness of an Access Token subsequently presented to this RS.

Another problem with clock drift is that evaluating the standard token expiration claim "exp" can give unpredictable results.

Acceptable ranges of clock drift are highly dependent on the concrete application. Important factors are how long access tokens are valid, and how critical timely expiration of access token is.

The expiration mechanism implemented by the "exi" claim, based on the first time the RS sees the token was defined to provide a more predictable alternative. The "exi" approach has some drawbacks that need to be considered:

A malicious client may hold back tokens with the "exi" claim in order to prolong their lifespan.

If an RS loses state (e.g. due to an unscheduled reboot), it may loose the current values of counters tracking the "exi" claims of tokens it is storing.

The first drawback is inherent to the deployment scenario and the "exi" solution. It can therefore not be mitigated without requiring the the RS be online at times. The second drawback can be mitigated by regularly storing the value of "exi" counters to persistent memory.

6.7. Combining profiles

There may be use cases were different profiles of this framework are combined. For example, an MQTT-TLS profile is used between the client and the RS in combination with a CoAP-DTLS profile for interactions between the client and the AS. The security of a profile MUST NOT depend on the assumption that the profile is used for all the different types of interactions in this framework.

6.8. Unprotected Information

Communication with the authz-info endpoint, as well as the various error responses defined in this framework, all potentially include sending information over an unprotected channel. These messages may leak information to an adversary, or may be manipulated by active attackers to induce incorrect behavior. For example error responses for requests to the Authorization Information endpoint can reveal information about an otherwise opaque access token to an adversary who has intercepted this token.

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

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

If the initial unauthorized resource request message (see Section 5.2) is used, the client MUST make sure that it is not sending sensitive content in this request. While GET and DELETE requests only reveal the target URI of the resource, POST and PUT requests would reveal the whole payload of the intended operation.

Since the client is not authenticated at the point when it is submitting an access token to the authz-info endpoint, attackers may be pretending to be a client and trying to trick an RS to use an obsolete profile that in turn specifies a vulnerable security mechanism via the authz-info endpoint. Such an attack would require a valid access token containing an "ace_profile" claim requesting the use of said obsolete profile. Resource Owners should update the configuration of their RS's to prevent them from using such obsolete profiles.

6.9. Identifying audiences

The audience claim as defined in [RFC7519] and the equivalent "audience" parameter from [RFC8693] are intentionally vague on how to match the audience value to a specific RS. This is intended to allow application specific semantics to be used. This section attempts to give some general guidance for the use of audiences in constrained environments.

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

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

Even the client must be able to determine the correct values to put into the "audience" parameter, in order to obtain a token for the intended RS. Errors in this process can lead to the client inadvertently obtaining a token for the wrong RS. The correct values for "audience" can either be provisioned to the client as part of its configuration, or dynamically looked up by the client in some directory. In the latter case the integrity and correctness of the directory data must be assured. Note that the "audience" hint provided by the RS as part of the "AS Request Creation Hints" Section 5.3 is not typically source authenticated and integrity protected, and should therefore not be treated a trusted value.

6.10. Denial of service against or with Introspection

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

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

Second, an attacker could use the fact that an RS performs introspection to perform a denial of service attack against that RS by repeatedly sending tokens to its authz-info endpoint that require an introspection call. RS can mitigate such attacks by implementing rate limits on how many introspection requests they perform in a given time interval for a certain client IP address submitting tokens

to /authz-info. When that limit has been reached, incoming requests from that address are rejected for a certain amount of time. A general rate limit on the introspection requests should also be considered, to mitigate distributed attacks.

7. Privacy Considerations

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

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

The claims contained in a token can reveal privacy sensitive information about the client and the RS to any party having access to them (whether by processing the content of a self-contained token or by introspection). The AS SHOULD be configured to minimize the information about clients and RSs disclosed in the tokens it issues.

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

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

An unprotected response to an unauthorized request (see Section 5.3) may disclose information about RS and/or its existing relationship with C. It is advisable to include as little information as possible in an unencrypted response. Even the absolute URI of the AS may reveal sensitive information about the service that RS provides. Developers must ensure that the RS does not disclose information that has an impact on the privacy of the stakeholders in the AS Request Creation Hints. They may choose to use a different mechanism for the

discovery of the AS if necessary. If means of encrypting communication between C and RS already exist, more detailed information may be included with an error response to provide C with sufficient information to react on that particular error.

8. IANA Considerations

This document creates several registries with a registration policy of "Expert Review"; guidelines to the experts are given in Section 8.17.

8.1. ACE Authorization Server Request Creation Hints

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

The columns of the registry are:

Name The name of the parameter

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

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

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

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

8.2. CoRE Resource Type registry

IANA is requested to register a new Resource Type (rt=) Link Target Attribute in the "Resource Type (rt=) Link Target Attribute Values" subregistry under the "Constrained RESTful Environments (CoRE) Parameters" [IANA.CoreParameters] registry:

rt="ace.ai". This resource type describes an ACE-OAuth authz-info endpoint resource.

Specific ACE-OAuth profiles can use this common resource type for defining their profile-specific discovery processes.

8.3. OAuth Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry [IANA.OAuthExtensionsErrorRegistry].

- Error name: "unsupported_pop_key"
- Error usage location: token error response
- Related protocol extension: [this document]
- Change Controller: IESG
- Specification document(s): Section 5.8.3 of [this document]

- Error name: "incompatible_ace_profiles"
- Error usage location: token error response
- Related protocol extension: [this document]
- Change Controller: IESG
- Specification document(s): Section 5.8.3 of [this document]

8.4. OAuth Error Code CBOR Mappings Registry

This specification establishes the IANA "OAuth Error Code CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of the registry are:

Name The OAuth Error Code name, refers to the name in Section 5.2. of [RFC6749], e.g., "invalid_request".
CBOR Value CBOR abbreviation for this error code. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference This contains a pointer to the public specification of the error code abbreviation, if one exists.

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

8.5. OAuth Grant Type CBOR Mappings

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

Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name The name of the grant type as specified in Section 1.3 of [RFC6749].
CBOR Value CBOR abbreviation for this grant type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference This contains a pointer to the public specification of the grant type abbreviation, if one exists.
Original Specification This contains a pointer to the public specification of the grant type, if one exists.

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

8.6. OAuth Access Token Types

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

- Type name: "PoP"
- Additional Token Endpoint Response Parameters: "cnf", "rs_cnf" see section 3.3 of [I-D.ietf-ace-oauth-params].
- HTTP Authentication Scheme(s): N/A
- Change Controller: IETF
- Specification document(s): [this document]

8.7. OAuth Access Token Type CBOR Mappings

This specification established the IANA "OAuth Access Token Type CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name The name of token type as registered in the OAuth Access Token Types registry, e.g., "Bearer".
CBOR Value CBOR abbreviation for this token type. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Reference This contains a pointer to the public specification of the OAuth token type abbreviation, if one exists.
Original Specification This contains a pointer to the public specification of the OAuth token type, if one exists.

8.7.1. Initial Registry Contents

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

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

8.8. ACE Profile Registry

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

The columns of this registry are:

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

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

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

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

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

8.9. OAuth Parameter Registration

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

- Name: "ace_profile"
- Parameter Usage Location: token response
- Change Controller: IESG
- Reference: Section 5.8.2 and Section 5.8.4.3 of [this document]

8.10. OAuth Parameters CBOR Mappings Registry

This specification establishes the IANA "OAuth Parameters CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".
CBOR Key CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Value Type The allowable CBOR data types for values of this parameter.
Reference This contains a pointer to the public specification of the OAuth parameter abbreviation, if one exists.

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

8.11. OAuth Introspection Response Parameter Registration

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

- Name: "ace_profile"
- Description: The ACE profile used between client and RS.
- Change Controller: IESG
- Reference: Section 5.9.2 of [this document]

- Name: "cnonce"
- Description: "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- Change Controller: IESG
- Reference: Section 5.9.2 of [this document]

- Name: "exi"
- Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker from of token expiration for devices that cannot synchronize their internal clocks.
- Change Controller: IESG
- Reference: Section 5.9.2 of [this document]

8.12. OAuth Token Introspection Response CBOR Mappings Registry

This specification establishes the IANA "OAuth Token Introspection Response CBOR Mappings" registry. The registry has been created to use the "Expert Review" registration procedure [RFC8126], except for the value range designated for private use.

The columns of this registry are:

Name The OAuth Parameter name, refers to the name in the OAuth parameter registry, e.g., "client_id".
CBOR Key CBOR map key for this parameter. Integer values less than -65536 are marked as "Private Use", all other values use the registration policy "Expert Review" [RFC8126].
Value Type The allowable CBOR data types for values of this parameter.
Reference This contains a pointer to the public specification of the introspection response parameter abbreviation, if one exists.

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

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

8.13. JSON Web Token Claims

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

- Claim Name: "ace_profile"
- Claim Description: The ACE profile a token is supposed to be used with.
- Change Controller: IESG
- Reference: Section 5.10 of [this document]

- Claim Name: "cnonce"
- Claim Description: "client-nonce". A nonce previously provided to the AS by the RS via the client. Used to verify token freshness when the RS cannot synchronize its clock with the AS.
- Change Controller: IESG
- Reference: Section 5.10 of [this document]

- Claim Name: "exi"
- Claim Description: "Expires in". Lifetime of the token in seconds from the time the RS first sees it. Used to implement a weaker

from of token expiration for devices that cannot synchronize their internal clocks.
- Change Controller: IESG
- Reference: Section 5.10.3 of [this document]

8.14. CBOR Web Token Claims

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

- Claim Name: "ace_profile"
- Claim Description: The ACE profile a token is supposed to be used with.
- JWT Claim Name: ace_profile
- Claim Key: TBD (suggested: 38)
- Claim Value Type(s): integer
- Change Controller: IESG
- Specification Document(s): Section 5.10 of [this document]

- Claim Name: "cnonce"
- Claim Description: The client-nonce sent to the AS by the RS via the client.
- JWT Claim Name: cnonce
- Claim Key: TBD (suggested: 39)
- Claim Value Type(s): byte string
- Change Controller: IESG
- Specification Document(s): Section 5.10 of [this document]

- Claim Name: "exi"
- Claim Description: The expiration time of a token measured from when it was received at the RS in seconds.
- JWT Claim Name: exi
- Claim Key: TBD (suggested: 40)
- Claim Value Type(s): integer
- Change Controller: IESG
- Specification Document(s): Section 5.10.3 of [this document]

- Claim Name: "scope"
- Claim Description: The scope of an access token as defined in [RFC6749].
- JWT Claim Name: scope
- Claim Key: TBD (suggested: 9)
- Claim Value Type(s): byte string or text string
- Change Controller: IESG
- Specification Document(s): Section 4.2 of [RFC8693]

8.15. Media Type Registrations

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

Type name: application

Subtype name: ace+cbor

Required parameters: N/A

Optional parameters: N/A

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

Security considerations: See Section 6 of [this document]

Interoperability considerations: N/A

Published specification: [this document]

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

Fragment identifier considerations: N/A

Additional information: N/A

Person & email address to contact for further information: <iesg@ietf.org>

Intended usage: COMMON

Restrictions on usage: none

Author: Ludwig Seitz <ludwig.seitz@combitech.se>

Change controller: IESG

8.16. CoAP Content-Format Registry

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

Media Type: application/ace+cbor

Encoding: -

ID: TBD (suggested: 19)

Reference: [this document]

8.17. Expert Review Instructions

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

Expert reviewers should take into consideration the following points:

- Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered, and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.
- Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on.
- Since a high degree of overlap is expected between these registries and the contents of the OAuth parameters [IANA.OAuthParameters] registries, experts should require new registrations to maintain alignment with parameters from OAuth that have comparable functionality. Deviation from this alignment should only be allowed if there are functional differences, that are motivated by the use case and that cannot be easily or efficiently addressed by comparable OAuth parameters.

9. Acknowledgments

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

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

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

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

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

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

Thanks to Cigdem Sengul for some very useful review comments.

Thanks to Carsten Bormann for contributing the text for the CoRE Resource Type registry.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova. Ludwig Seitz was also received further funding for this work by Vinnova in the context of the CelticNext project Critisec.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-params]
Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-13 (work in progress), April 2020.

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

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

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

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

[IANA.OAuthExtensionsErrorRegistry]
IANA, "OAuth Extensions Error Registry",
<https://www.iana.org/assignments/oauth-parameters/oauth-parameters.xhtml#extensions-error>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/info/rfc8693>.

[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>.

10.2. Informative References

[BLE] Bluetooth SIG, "Bluetooth Core Specification v5.1", Section 4.4, January 2019, <https://www.bluetooth.com/specifications/bluetooth-core-specification/>.

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

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-34 (work in progress), January 2021.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-40 (work in progress), January 2021.

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

[MQTT5.0] Banks, A., Briggs, E., Borgendale, K., and R. Gupta, "MQTT Version 5.0", OASIS Standard, March 2019, <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

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

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

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

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

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

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

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

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

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

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

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

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

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

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

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

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

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/info/rfc8628>.

Appendix A. Design Justification

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

Common IoT constraints are:

Low Power Radio:

Many IoT devices are equipped with a small battery which needs to last for a long time. For many constrained wireless devices, the highest energy cost is associated to transmitting or receiving messages (roughly by a factor of 10 compared to AES) [Margi10impact]. It is therefore important to keep the total communication overhead low, including minimizing the number and size of messages sent and received, which has an impact of choice on the message format and protocol. By using CoAP over UDP and CBOR encoded messages, some of these aspects are addressed. Security protocols contribute to the communication overhead and can, in some cases, be optimized. For example, authentication and key establishment may, in certain cases where security requirements allow, be replaced by provisioning of security context by a trusted third party, using transport or application layer security.

Low CPU Speed:

Some IoT devices are equipped with processors that are significantly slower than those found in most current devices on the Internet. This typically has implications on what timely cryptographic operations a device is capable of performing, which in turn impacts, e.g., protocol latency. Symmetric key cryptography may be used instead of the computationally more expensive public key cryptography where the security requirements so allow, but this may also require support for trusted-third-party-assisted secret key establishment using transport- or application-layer security.

Small Amount of Memory:

Microcontrollers embedded in IoT devices are often equipped with only a small amount of RAM and flash memory, which places limitations on what kind of processing can be performed and how much code can be put on those devices. To reduce code size, fewer and smaller protocol implementations can be put on the firmware of such a device. In this case, CoAP may be used instead of HTTP,

symmetric-key cryptography instead of public-key cryptography, and CBOR instead of JSON. An authentication and key establishment protocol, e.g., the DTLS handshake, in comparison with assisted key establishment, also has an impact on memory and code footprints.

User Interface Limitations:

Protecting access to resources is both an important security as well as privacy feature. End users and enterprise customers may not want to give access to the data collected by their IoT device or to functions it may offer to third parties. Since the classical approach of requesting permissions from end users via a rich user interface does not work in many IoT deployment scenarios, these functions need to be delegated to user-controlled devices that are better suitable for such tasks, such as smart phones and tablets.

Communication Constraints:

In certain constrained settings an IoT device may not be able to communicate with a given device at all times. Devices may be sleeping, or just disconnected from the Internet because of general lack of connectivity in the area, for cost reasons, or for security reasons, e.g., to avoid an entry point for Denial-of-Service attacks.

The communication interactions this framework builds upon (as shown graphically in Figure 1) may be accomplished using a variety of different protocols, and not all parts of the message flow are used in all applications due to the communication constraints. Deployments making use of CoAP are expected, but this framework is not limited to them. Other protocols such as HTTP, or even protocols such as Bluetooth Smart communication that do not necessarily use IP, could also be used. The latter raises the need for application layer security over the various interfaces.

In the light of these constraints we have made the following design decisions:

CBOR, COSE, CWT:

This framework RECOMMENDS the use of CBOR [RFC8949] as data format. Where CBOR data needs to be protected, the use of COSE [RFC8152] is RECOMMENDED. Furthermore, where self-contained tokens are needed, this framework RECOMMENDS the use of CWT [RFC8392]. These measures aim at reducing the size of messages sent over the wire, the RAM size of data objects that need to be

kept in memory and the size of libraries that devices need to support.

CoAP:

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

Access Information:

This framework defines the name "Access Information" for data concerning the RS that the AS returns to the client in an access token response (see Section 5.8.2). This aims at enabling scenarios where a powerful client, supporting multiple profiles, needs to interact with an RS for which it does not know the supported profiles and the raw public key.

Proof-of-Possession:

This framework makes use of proof-of-possession tokens, using the "cnf" claim [RFC8747]. A request parameter "cnf" and a Response parameter "cnf", both having a value space semantically and syntactically identical to the "cnf" claim, are defined for the token endpoint, to allow requesting and stating confirmation keys. This aims at making token theft harder. Token theft is specifically relevant in constrained use cases, as communication often passes through middle-boxes, which could be able to steal bearer tokens and use them to gain unauthorized access.

Authz-Info endpoint:

This framework introduces a new way of providing access tokens to an RS by exposing a authz-info endpoint, to which access tokens can be POSTed. This aims at reducing the size of the request message and the code complexity at the RS. The size of the request message is problematic, since many constrained protocols have severe message size limitations at the physical layer (e.g., in the order of 100 bytes). This means that larger packets get fragmented, which in turn combines badly with the high rate of packet loss, and the need to retransmit the whole message if one packet gets lost. Thus separating sending of the request and sending of the access tokens helps to reduce fragmentation.

Client Credentials Grant:

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

Introspection:

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

Appendix B. Roles and Responsibilities

Resource Owner

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

Requesting Party

* Make sure that the client is provisioned the necessary credentials to authenticate to the AS.
* Make sure that the client is configured to follow the security requirements of the Requesting Party when issuing requests

(e.g., minimum communication security requirements, trust anchors).
* Register the client at the AS. This includes making known to the AS which profiles, token_types, and key types (symmetric/ asymmetric) the client.

Authorization Server

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

Client

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

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

Resource Server

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

    + Verify the token is from a recognized AS.
    + Check the token's integrity.
    + Verify that the token applies to this RS.
    + Check that the token has not expired (if the token provides expiration information).
    + Store the token so that it can be retrieved in the context of a matching request.

    Note: The order proposed here is not normative, any process that arrives at an equivalent result can be used. A noteworthy consideration is whether one can use cheap operations early on to quickly discard non-applicable or invalid tokens, before performing expensive cryptographic operations (e.g. doing an expiration check before verifying a signature).

  * Process a request.

    + Set up communication security with the client.
    + Authenticate the client.
    + Match the client against existing tokens.
    + Check that tokens belonging to the client actually authorize the requested action.
    + Optionally: Check that the matching tokens are still valid, using introspection (if this is possible.)
  * Send a response following the agreed upon communication security mechanism(s).
  * Safely store credentials such as raw public keys for authentication or proof-of-possession keys linked to access tokens.

Appendix C. Requirements on Profiles

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

- Optionally define new methods for the client to discover the necessary permissions and AS for accessing a resource, different from the one proposed in Section 5.1. Section 4
- Optionally specify new grant types. Section 5.4
- Optionally define the use of client certificates as client credential type. Section 5.5
- Specify the communication protocol the client and RS the must use (e.g., CoAP). Section 5 and Section 5.8.4.3
- Specify the security protocol the client and RS must use to protect their communication (e.g., OSCORE or DTLS). This must provide encryption, integrity and replay protection. Section 5.8.4.3
- Specify how the client and the RS mutually authenticate. Section 4
- Specify the proof-of-possession protocol(s) and how to select one, if several are available. Also specify which key types (e.g., symmetric/asymmetric) are supported by a specific proof-of-possession protocol. Section 5.8.4.2
- Specify a unique ace_profile identifier. Section 5.8.4.3
- If introspection is supported: Specify the communication and security protocol for introspection. Section 5.9
- Specify the communication and security protocol for interactions between client and AS. This must provide encryption, integrity protection, replay protection and a binding between requests and responses. Section 5 and Section 5.8
- Specify how/if the authz-info endpoint is protected, including how error responses are protected. Section 5.10.1
- Optionally define other methods of token transport than the authz-info endpoint. Section 5.10.1

Appendix D. Assumptions on AS knowledge about C and RS

This section lists the assumptions on what an AS should know about a client and an RS in order to be able to respond to requests to the token and introspection endpoints. How this information is established is out of scope for this document.

- The identifier of the client or RS.
- The profiles that the client or RS supports.
- The scopes that the RS supports.
- The audiences that the RS identifies with.
- The key types (e.g., pre-shared symmetric key, raw public key, key length, other key parameters) that the client or RS supports.

- The types of access tokens the RS supports (e.g., CWT).
- If the RS supports CWTs, the COSE parameters for the crypto wrapper (e.g., algorithm, key-wrap algorithm, key-length) that the RS supports.
- The expiration time for access tokens issued to this RS (unless the RS accepts a default time chosen by the AS).
- The symmetric key shared between client and AS (if any).
- The symmetric key shared between RS and AS (if any).
- The raw public key of the client or RS (if any).
- Whether the RS has synchronized time (and thus is able to use the 'exp' claim) or not.

Appendix E. Deployment Examples

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

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

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

E.1. Local Token Validation

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

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

This example shows the interactions between a client, the authorization server and a temperature sensor acting as a resource server. Message exchanges A and B are shown in Figure 17.

A: The client first generates a public-private key pair used for communication security with the RS.
The client sends a CoAP POST request to the token endpoint at the AS. The security of this request can be transport or application layer. It is up the the communication security profile to define.

In the example it is assumed that both client and AS have performed mutual authentication e.g. via DTLS. The request contains the public key of the client and the Audience parameter set to "tempSensorInLivingRoom", a value that the temperature sensor identifies itself with. The AS evaluates the request and authorizes the client to access the resource.
B: The AS responds with a 2.05 Content response containing the Access Information, including the access token. The PoP access token contains the public key of the client, and the Access Information contains the public key of the RS. For communication security this example uses DTLS RawPublicKey between the client and the RS. The issued token will have a short validity time, i.e., "exp" close to "iat", in order to mitigate attacks using stolen client credentials. The token includes the claim such as "scope" with the authorized access that an owner of the temperature device can enjoy. In this example, the "scope" claim, issued by the AS, informs the RS that the owner of the token, that can prove the possession of a key is authorized to make a GET request against the /temperature resource and a POST request on the /firmware resource. Note that the syntax and semantics of the scope claim are application specific.
Note: In this example it is assumed that the client knows what resource it wants to access, and is therefore able to request specific audience and scope claims for the access token.

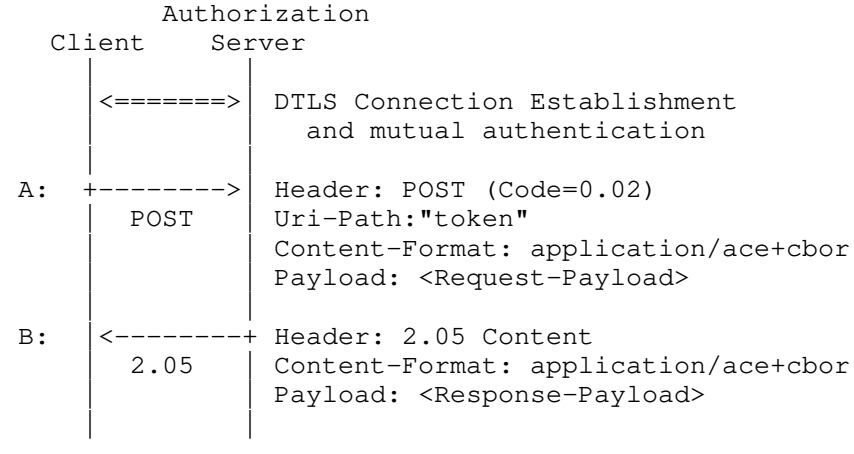


Figure 17: Token Request and Response Using Client Credentials.

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

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

Response-Payload :
{
  "access_token" : b64'0INDoQEKoQVNKkXfb7xaWqMTf6 ...',
  "rs_cnf" : {
    "COSE_Key" : {
      "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk',
      "kty" : "EC",
      "crv" : "P-256",
      "x"   : b64'MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4',
      "y"   : b64'4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM'
    }
  }
}
```

Figure 18: Request and Response Payload Details.

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

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1563451500",
  "exp" : "1563453000",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "COSE_Key" : {
      "kid" : b64'1Bg8vub9tLe1gHMzV76e8',
      "kty" : "EC",
      "crv" : "P-256",
      "x" : b64'f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU',
      "y" : b64'x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0'
    }
  }
}
```

Figure 19: Access Token including Public Key of the Client.

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

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

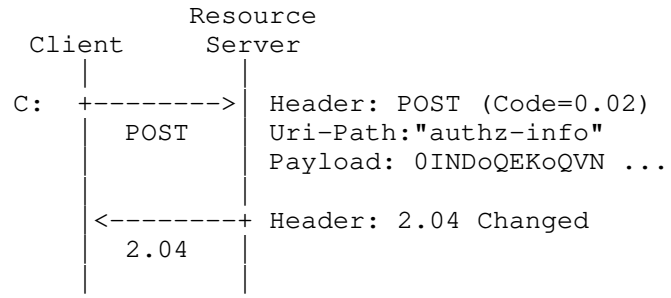


Figure 20: Access Token provisioning to RS

The client and the RS runs the DTLS handshake using the raw public keys established in step B and C.
The client sends a CoAP GET request to /temperature on RS over DTLS. The RS verifies that the request is authorized, based on previously established security context.

F: The RS responds over the same DTLS channel with a CoAP 2.05 Content response, containing a resource representation as payload.

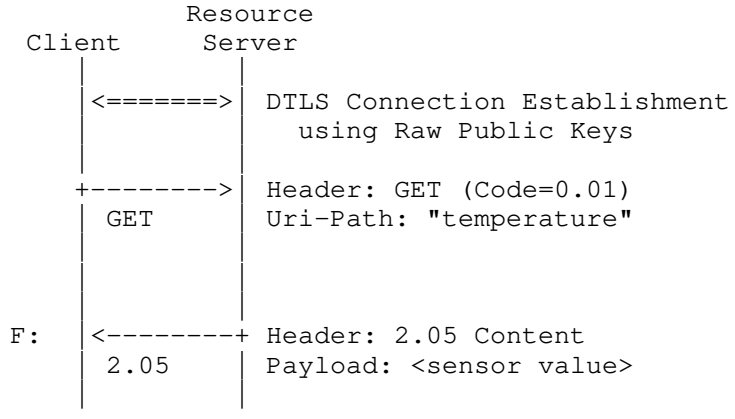


Figure 21: Resource Request and Response protected by DTLS.

E.2. Introspection Aided Token Validation

In this deployment scenario it is assumed that a client is not able to access the AS at the time of the access request, whereas the RS is assumed to be connected to the back-end infrastructure. Thus the RS can make use of token introspection. This access procedure involves steps A-F of Figure 1, but assumes steps A and B have been carried out during a phase when the client had connectivity to AS.

Since the client is assumed to be offline, at least for a certain period of time, a pre-provisioned access token has to be long-lived. Since the client is constrained, the token will not be self contained (i.e. not a CWT) but instead just a reference. The resource server uses its connectivity to learn about the claims associated to the access token by using introspection, which is shown in the example below.

In the example interactions between an offline client (key fob), an RS (online lock), and an AS is shown. It is assumed that there is a provisioning step where the client has access to the AS. This corresponds to message exchanges A and B which are shown in Figure 22.

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

to his computer that then provides the UI for the device. After that OAuth 2.0 implicit flow can used to authorize the key for his car at the the car manufacturers AS.

Note: In this example the client does not know the exact door it will be used to access since the token request is not send at the time of access. So the scope and audience parameters are set quite wide to start with, while tailored values narrowing down the claims to the specific RS being accessed can be provided to that RS during an introspection step.

A: The client sends a CoAP POST request to the token endpoint at AS. The request contains the Audience parameter set to "PACS1337" (PACS, Physical Access System), a value the that identifies the physical access control system to which the individual doors are connected. The AS generates an access token as an opaque string, which it can match to the specific client and the targeted audience. It furthermore generates a symmetric proof-of-possession key. The communication security and authentication between client and AS is assumed to have been provided at transport layer (e.g. via DTLS) using a pre-shared security context (psk, rpk or certificate).
B: The AS responds with a CoAP 2.05 Content response, containing as playload the Access Information, including the access token and the symmetric proof-of-possession key. Communication security between C and RS will be DTLS and PreSharedKey. The PoP key is used as the PreSharedKey.

Note: In this example we are using a symmetric key for a multi-RS audience, which is not recommended normally (see Section 6.9). However in this case the risk is deemed to be acceptable, since all the doors are part of the same physical access control system, and therefore the risk of a malicious RS impersonating the client towards another RS is low.

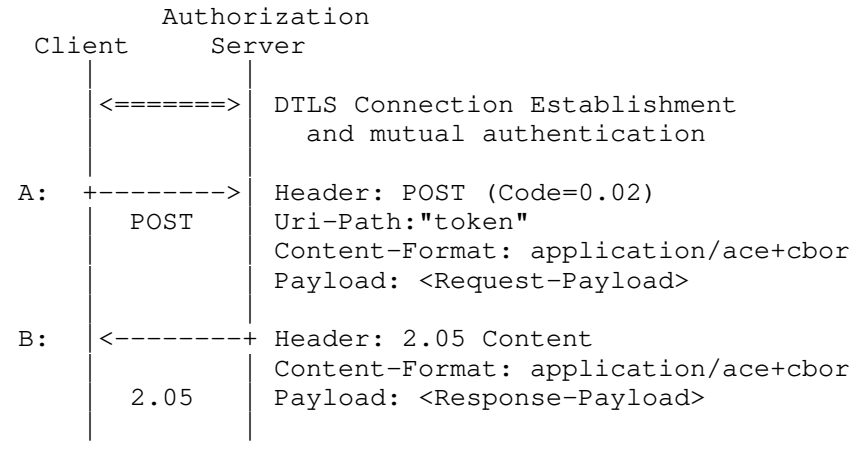


Figure 22: Token Request and Response using Client Credentials.

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

```
Request-Payload:
{
  "client_id" : "keyfob",
  "audience" : "PACS1337"
}

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

Figure 23: Request and Response Payload for C offline

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

C: Next, the client POSTs the access token to the authz-info endpoint in the RS. This is a plain CoAP request, i.e., no DTLS between client and RS. Since the token is an opaque string, the

RS cannot verify it on its own, and thus defers to respond the client with a status code until after step E.
D: The RS sends the token to the introspection endpoint on the AS using a CoAP POST request. In this example RS and AS are assumed to have performed mutual authentication using a pre shared security context (psk, rpk or certificate) with the RS acting as DTLS client.
E: The AS provides the introspection response (2.05 Content) containing parameters about the token. This includes the confirmation key (cnf) parameter that allows the RS to verify the client's proof of possession in step F. Note that our example in Figure 25 assumes a pre-established key (e.g. one used by the client and the RS for a previous token) that is now only referenced by its key-identifier 'kid'.
After receiving message E, the RS responds to the client's POST in step C with the CoAP response code 2.01 (Created).

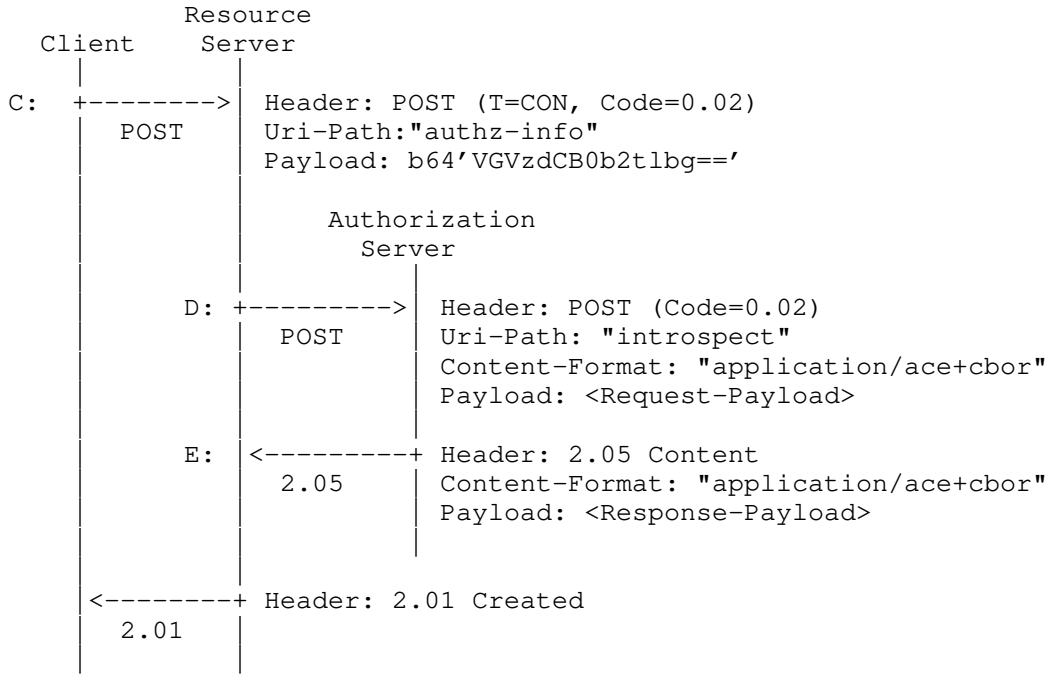


Figure 24: Token Introspection for C offline

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

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

Response-Payload:
{
  "active" : true,
  "aud" : "lockOfDoor4711",
  "scope" : "open, close",
  "iat" : 1563454000,
  "cnf" : {
    "kid" : b64'c29tZSBwdWJsaWMga2V5IGlk'
  }
}
```

Figure 25: Request and Response Payload for Introspection

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

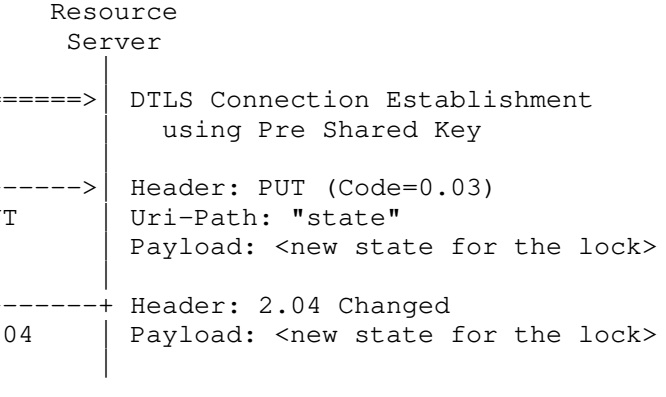


Figure 26: Resource request and response protected by OSCORE

Appendix F. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

F.1. Version -21 to 22

- Provided section numbers in references to OAuth RFC.
- Updated IANA mapping registries to only use "Private Use" and "Expert Review".
- Made error messages optional for RS at token submission since it may not be able to send them depending on the profile.
- Corrected errors in examples.

F.2. Version -20 to 21

- Added text about expiration of RS keys.

F.3. Version -19 to 20

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

F.4. Version -18 to -19

- Added definition of "Authorization Information".
- Explicitly state that ACE allows encoding refresh tokens in binary format in addition to strings.
- Renamed "AS Information" to "AS Request Creation Hints" and added the possibility to specify req_aud and scope as hints.
- Added the "kid" parameter to AS Request Creation Hints.
- Added security considerations about the integrity protection of tokens with multi-RS audiences.
- Renamed IANA registries mapping OAuth parameters to reflect the mapped registry.
- Added JWT claim names to CWT claim registrations.
- Added expert review instructions.
- Updated references to TLS from 1.2 to 1.3.

F.5. Version -17 to -18

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

F.6. Version -16 to -17

- Added references to (D)TLS 1.3.
- Added requirement that responses are bound to requests.

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

F.7. Version -15 to -16

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

F.8. Version -14 to -15

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

F.9. Version -13 to -14

- Split out the 'aud', 'cnf' and 'rs_cnf' parameters to [I-D.ietf-ace-oauth-params]
- Introduced the "application/ace+cbor" Content-Type.
- Added claim registrations from 'profile' and 'rs_cnf'.
- Added note on schema part of AS Information Section 5.3
- Realigned the parameter abbreviations to push rarely used ones to the 2-byte encoding size of CBOR integers.

F.10. Version -12 to -13

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

F.11. Version -11 to -12

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

F.12. Version -10 to -11

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

F.13. Version -09 to -10

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

F.14. Version -08 to -09

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

F.15. Version -07 to -08

- Moved AS discovery from the DTLS profile to the framework, see Section 5.1.
- Made the use of CBOR mandatory. If you use JSON you can use vanilla OAuth.
- Made it mandatory for profiles to specify C-AS security and RS-AS security (the latter only if introspection is supported).
- Made the use of CBOR abbreviations mandatory.

- Added text to clarify the use of token references as an alternative to CWTs.
- Added text to clarify that introspection must not be delayed, in case the RS has to return a client token.
- Added security considerations about leakage through unprotected AS discovery information, combining profiles and leakage through error responses.
- Added privacy considerations about leakage through unprotected AS discovery.
- Added text that clarifies that introspection is optional.
- Made profile parameter optional since it can be implicit.
- Clarified that CoAP is not mandatory and other protocols can be used.
- Clarified the design justification for specific features of the framework in appendix A.
- Clarified appendix E.2.
- Removed specification of the "cnf" claim for CBOR/COSE, and replaced with references to [RFC8747]

F.16. Version -06 to -07

- Various clarifications added.
- Fixed erroneous author email.

F.17. Version -05 to -06

- Moved sections that define the ACE framework into a subsection of the framework Section 5.
- Split section on client credentials and grant into two separate sections, Section 5.4, and Section 5.5.
- Added Section 5.6 on AS authentication.
- Added Section 5.7 on the Authorization endpoint.

F.18. Version -04 to -05

- Added RFC 2119 language to the specification of the required behavior of profile specifications.
- Added Section 5.5 on the relation to the OAuth2 grant types.
- Added CBOR abbreviations for error and the error codes defined in OAuth2.
- Added clarification about token expiration and long-running requests in Section 5.10.3
- Added security considerations about tokens with symmetric PoP keys valid for more than one RS.
- Added privacy considerations section.
- Added IANA registry mapping the confirmation types from RFC 7800 to equivalent COSE types.

- Added appendix D, describing assumptions about what the AS knows about the client and the RS.

F.19. Version -03 to -04

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

F.20. Version -02 to -03

- Removed references to draft-ietf-oauth-pop-key-distribution since the status of this draft is unclear.
- Copied and adapted security considerations from draft-ietf-oauth-pop-key-distribution.
- Renamed "client information" to "RS information" since it is information about the RS.
- Clarified the requirements on profiles of this framework.
- Clarified the token endpoint protocol and removed negotiation of "profile" and "alg" (section 6).
- Renumbered the abbreviations for claims and parameters to get a consistent numbering across different endpoints.
- Clarified the introspection endpoint.
- Renamed token, introspection and authz-info to "endpoint" instead of "resource" to mirror the OAuth 2.0 terminology.
- Updated the examples in the appendices.

F.21. Version -01 to -02

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

F.22. Version -00 to -01

- Changed 5.1. from "Communication Security Protocol" to "Client Information".
- Major rewrite of 5.1 to clarify the information exchanged between C and AS in the PoP access token request profile for IoT.
  - Allow the client to indicate preferences for the communication security protocol.
  - Defined the term "Client Information" for the additional information returned to the client in addition to the access token.
  - Require that the messages between AS and client are secured, either with (D)TLS or with COSE_Encrypted wrappers.
  - Removed dependency on OSCOAP and added generic text about object security instead.
  - Defined the "rpk" parameter in the client information to transmit the raw public key of the RS from AS to client.
  - (D)TLS MUST use the PoP key in the handshake (either as PSK or as client RPK with client authentication).
  - Defined the use of x5c, x5t and x5tS256 parameters when a client certificate is used for proof of possession.
  - Defined "tktn" parameter for signaling for how to transfer the access token.
- Added 5.2. the CoAP Access-Token option for transferring access tokens in messages that do not have payload.
- 5.3.2. Defined success and error responses from the RS when receiving an access token.
- 5.6.:Added section giving guidance on how to handle token expiration in the absence of reliable time.
- Appendix B Added list of roles and responsibilities for C, AS and RS.

Authors' Addresses

Ludwig Seitz
Combitech
Djaeknegatan 31
Malmoe 211 35
Sweden

Email: ludwig.seitz@combitech.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Erik Wahlstroem
Sweden

Email: erik@wahlstromstekniska.se

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

Email: erdtman@spotify.com

Hannes Tschofenig
Arm Ltd.
Absam 6067
Austria

Email: Hannes.Tschofenig@arm.com

ACE Working Group
Internet-Draft
Intended Status: Standards Track
Expires: January 2, 2017

L. Seitz
SICS Swedish ICT

July 1, 2016

OSCOAP profile of ACE
draft-seitz-ace-ocsoap-profile-00

Abstract

This memo specifies a profile for the ACE framework for Authentication and Authorization. It utilizes Object Security of CoAP (OSCOAP) and Ephemeral Diffie-Hellman over COSE (EDHOC) to provide communication security, server authentication, and proof-of-possession for a key owned by the client and bound to an OAuth 2.0 access token.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Table of Contents

1. Introduction

This memo specifies a profile of the ACE framework [I-D.ietf-ace-oauth-authz]. In this profile, a client and a resource server use CoAP to communicate. The client uses an access token, bound to a key (the proof-of-possession key) to authorize its access to the resource server. In order to provide communication security, proof of possession, and server authentication they use Object Security of CoAP (OSCOAP) [I-D.selander-ace-object-security] and Ephemeral Diffie-Hellman Over COSE (EDHOC) [I-D.selander-ace-cose-ecdhe]. Optionally the client and the resource server may also use CoAP and OSCOAP to communicate with the authorization server. The use of EDHOC in this profile in addition to OSCOAP, provides perfect forward secrecy (PFS) and the initial proof-of-possession, which ties the proof-of-possession key to an OSCOAP security context.

OSCOAP specifies how to use CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg] to secure CoAP messages. In order to provide replay and reordering protection OSCOAP also introduces sequence numbers that are used together with COSE. EDHOC specifies an authenticated Diffie-Hellman protocol that allows two parties that wish to use OSCOAP to establish a shared secret key with perfect forward secrecy.

1.1 Terminology

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

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

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

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

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

2. Client to Resource Server

The use of OSCOAP for arbitrary CoAP messages is specified in [I-D.selander-ace-object-security]. This section defines the specific uses and their purpose for securing the communication between a client and a resource server, and the parameters for to negotiating the use of this profile with the token endpoint at the authorization server as specified in section 6 of the ACE framework [I-D.ietf-ace-oauth-authz].

2.1. Signalling the use of OSCOAP

A client requesting a token at an AS via the /token endpoint MAY signal a preference for using OSCOAP by including the "profile" parameter with the value "coap_oscoap" in it’s access token request. This follows the message formats specified in section 6.1 of the ACE framework.

The AS responding to a successful access token request as defined in section 6.2 of the ACE framework can signal that the use of OSCOAP is REQUIRED for a specific access token by including the "profile" parameter with the value "coap_oscoap" in the access token response. This means that the client MUST use OSCOAP towards all resource servers for which this access token is valid.

The error response procedures defined in section 6.3 of the ACE framework are unchanged by this profile.

Note the the client and the authorization server MAY OPTIONALLY use OSCOAP to protect the interaction via the /token endpoint. See section 3 for details.

2.2. Key establishment for OSCOAP

Section 3.2 of OSCOAP [I-D.selander-ace-object-security] defines how to derive a security context based on a pre-shared secret established between client and server. If the proof-of-possession key is a symmetric key, it MAY be directly used as shared secret with OSCOAP.

However to provide forward secrecy and mutual authentication in the case of pre-established raw public keys or with X.509 certificates it is RECOMMENDED to use EDHOC [I-D.selander-ace-cose-ecdhe] to generate the initial shared key. EDHOC MUST be used as follows:

When the client sends the access token to the RS using the /authz-info endpoint as specified in section 8.1 of the ACE framework, this message MUST carry message_1 of the EDHOC protocol in the CoAP payload, and the access token MUST be included in the COSE unprotected header of message_1 as a CBOR map with the key

'access_token'.

When the RS responds to this token submission request, if the access token was valid the payload of the CoAP response MUST contain message_2 of the EDHOC protocol. If the token was not valid, the error response defined in the ACE framework is not modified. If the EDHOC message_1 was not valid the RS MUST respond with error code 4.01 (Unauthorized).

In the case of EDHOC being used with symmetric pop-keys, the protocol in section 3.4 of [I-D.selander-ace-cose-ecdhe] MUST be used. If the pop-key is asymmetric, the RS MUST also use an asymmetric key for authentication. This key is known to the client through the access token response (see section 6.2 of the ACE framework). In this case the protocol in section 3.5 of [I-D.selander-ace-cose-ecdhe] MUST be used.

Note that if the OSCOAP profile is used, the /authz-info endpoint at the Resource Server MUST be prepared to process and generate the protocol messages of the EDHOC protocol as specified above. Hence the use of EDHOC does not add any additional roundtrips to the ACE message exchange.

Figure 1 illustrates the message exchanges for using EDHOC on the authz-info endpoint.

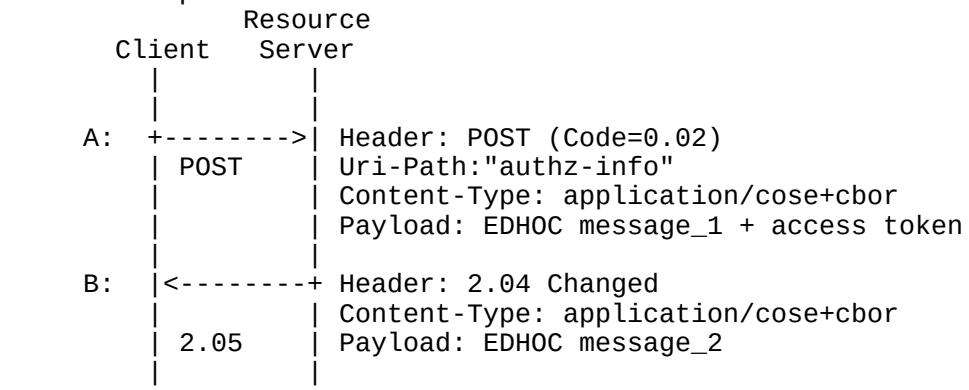


Figure 1: Key establishment with EDHOC via the authz-info endpoint

Figure 2 shows an example of message_1 with an access token embedded in the unprotected header.

```
997(
  [
     / protected / h'a201260444c150d41c',
         / 'alg' : 'ES256', 'kid' : 'kid_c' /
```

```
            / unprotected / {'access_token' : h'4a5015df6864286979'},
            / payload / h'83381a0c582fa120a501020241032001215820 98f
            50a4ff6c05861c8860d13a638ea56c3f5ad7590bbfbf054e1c7b4d9
            1d628022f5',
            / signature / h'eae868ecc1276883766c5dc5ba5b8dca25dab3c
            2e56a51ce5705b793914348e14eea4aee6e0c9f09db4ef3ddeca8f3
            506cd1a98a8fb64327be470355c9657ce0'
          ]
        )
```

Figure 2: EDHOC message_1 with an access token

2.3. Securing the Resource Request

When the client wishes to send a request to the RS, it uses the steps defined in section 6 of OSCOAP [I-D.selander-ace-object-security] to generate an OSCOAP message out of the unsecured CoAP message.

2.4. Securing the Resource Server Response

When a RS responds to a client's request, it uses the steps defined in section 6 of OSCOAP [I-D.selander-ace-object-security] to generate an OSCOAP message out of the unsecured CoAP message.

3. Client to Authorization Server

As specified in the ACE framework section 5 [I-D.ietf-ace-oauth-authz], the Client and AS can also use CoAP instead of HTTP to communicate via the token endpoint. This section specifies how to use OSCOAP between Client and AS together with CoAP. The use of OSCOAP for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The client and the AS are expected to have pre-established credentials (e.g. raw public keys). How these credentials are established is out of scope for this profile. Furthermore the client and the AS communicate using CoAP through the token endpoint as specified in section 6 of [I-D.ietf-ace-oauth-authz]. At first point of contact, prior to making the token request and response, the client and the AS MUST perform an EDHOC exchange with the pre-established credentials to create forward secret keying material for use with OSCOAP. Subsequent requests and the responses MUST be protected with OSCOAP.

4. Resource Server to Authorization Server

As specified in the ACE framework section 5 [I-D.ietf-ace-oauth-

authz], the RS and AS can also use CoAP instead of HTTP to communicate via the introspection endpoint. This section specifies how to use OSCOAP between RS and AS together with CoAP. The use of OSCOAP for this communication is OPTIONAL in this profile, other security protocols (such as DTLS) MAY be used instead.

The RS and the AS are expected to have pre-established credentials (e.g. symmetric keys). How these credentials are established is out of scope for this profile. Furthermore the RS and the AS communicate using CoAP through the introspection endpoint as specified in section 7 of [I-D.ietf-ace-oauth-authz]. At first point of contact, prior to making the introspection request and response, the RS and the AS MUST perform an EDHOC exchange with the pre-established credentials to create forward secret keying material for use with OSCOAP. Subsequent requests and the responses MUST be protected with OSCOAP.

5. Security Considerations

TBD.

6. Privacy Considerations

TBD.

7. IANA Considerations

FIXME: PoP alg: OSCOAP

8. Acknowledgements

The author wishes to thank Goeran Selander and Francesca Palombini for the input on this memo.

9. References

9.1 Normative References

[I-D.selander-ace-object-security] Selander, G., Mattsson J., Palombini F., and L. Seitz. "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-04 (work in progress), March 2016.

[I-D.selander-ace-cose-ecdhe] Selander, G., Mattsson J.,

and F. Palombini. "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-02 (work in progress), June 2016.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtmann, S., and H. Tschofenig. "Authentication and Authorization for Constrained Environments (ACE)", drart-ietf-ace-oauth-authz-02 (work in progress), June 2016.

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

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

9.2 Informative References

[I-D.gerdes-ace-actors]
Gerdes, S., Seitz, L., G. Selander, and C. Bormann (ed). "An Arhitecture for Authorization in Constrained Environments", draft-ietf-ace-actors-03 (work in progress), March 2016.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-14 (work in progress), June 2016.

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

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

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

Author's Address

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN
EMail: ludwig@sics.se

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2020

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

September 11, 2019

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-14

Abstract

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a very compact, and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys. EDHOC provides mutual authentication, perfect forward secrecy, and identity protection. EDHOC is intended for usage in constrained scenarios and a main use case is to establish an OSCORE security context. By reusing COSE for cryptography, CBOR for encoding, and CoAP for transport, the additional code footprint can be kept very low.

Table of Contents

1. Introduction

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs] or where the protection needs to work over a variety of underlying protocols. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and energy [RFC7228]. A method for protecting individual messages at the application layer suitable for constrained devices, is provided by CBOR Object Signing and Encryption (COSE) [RFC8152]), which builds on the Concise Binary Object Representation (CBOR) [I-D.ietf-cbor-7049bis]. Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] is a method for application-layer protection of the Constrained Application Protocol (CoAP), using COSE.

In order for a communication session to provide forward secrecy, the communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol with ephemeral keys, from which shared key material can be derived. This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a lightweight key exchange protocol providing perfect forward secrecy and identity protection. Authentication is based on credentials established out of band, e.g. from a trusted third party, such as an Authorization Server as specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports authentication using pre-shared keys (PSK), raw public keys (RPK), and public key certificates. After successful completion of the EDHOC protocol, application keys and other application specific data can be derived using the EDHOC-Exporter interface. A main use case for EDHOC is to establish an OSCORE security context. EDHOC uses COSE for cryptography, CBOR for encoding, and CoAP for transport. By reusing existing libraries, the additional code footprint can be kept very low. Note that this document focuses on authentication and key establishment: for integration with authorization of resource access, refer to [I-D.ietf-ace-oscore-profile].

EDHOC is designed to work in highly constrained scenarios making it especially suitable for network technologies such as Cellular IoT, 6TiSCH [I-D.ietf-6tisch-dtsecurity-zerotouch-join], and LoRaWAN [LoRa1][LoRa2]. These network technologies are characterized by their low throughput, low power consumption, and small frame sizes. Compared to the DTLS 1.3 handshake [I-D.ietf-tls-dtls13] with ECDH and connection ID, the number of bytes in EDHOC is less than 1/4 when PSK authentication is used and less than 1/3 when RPK authentication is used, see [I-D.ietf-lwig-security-protocol-comparison]. Typical message sizes for EDHOC with pre-shared keys, raw public keys, and X.509 certificates are shown in Figure 1.

	PSK	RPK	x5t	x5chain
message_1	40	38	38	38
message_2	45	114	126	116 + Certificate chain
message_3	11	80	91	81 + Certificate chain
Total	96	232	255	235 + Certificate chains

Figure 1: Typical message sizes in bytes

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56A [SP-800-56A], and HKDF [RFC5869]. CBOR [I-D.ietf-cbor-7049bis] and COSE [RFC8152] are used to implement these standards. The use of COSE provides crypto agility and enables use of future algorithms and headers designed for constrained IoT.

This document is organized as follows: Section 2 describes how EDHOC builds on SIGMA-I, Section 3 specifies general properties of EDHOC, including message flow, formatting of the ephemeral public keys, and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, Section 6 specifies the EDHOC error message, and Section 7 describes how EDHOC can be transferred in CoAP and used to establish an OSCORE security context.

1.1. Rationale for EDHOC

Many constrained IoT systems today do not use any security at all, and when they do, they often do not follow best practices. One reason is that many current security protocols are not designed with constrained IoT in mind. Constrained IoT systems often deal with personal information, valuable business data, and actuators interacting with the physical world. Not only do such systems need security and privacy, they often need end-to-end protection with

source authentication and perfect forward secrecy. EDHOC and OSCORE [RFC8613] enables security following current best practices to devices and systems where current security protocols are impractical.

EDHOC is optimized for small message sizes and can therefore be sent over a small number of radio frames. The message size of a key exchange protocol may have a large impact on the performance of an IoT deployment, especially in noisy environments. For example, in a network bootstrapping setting a large number of devices turned on in a short period of time may result in large latencies caused by parallel key exchanges. Requirements on network formation time in constrained environments can be translated into key exchange overhead. In networks technologies with transmission back-off time, each additional frame significantly increases the latency even if no other devices are transmitting.

Power consumption for wireless devices is highly dependent on message transmission, listening, and reception. For devices that only send a few bytes occasionally, the battery lifetime may be significantly reduced by a heavy key exchange protocol. Moreover, a key exchange may need to be executed more than once, e.g. due to a device losing power or rebooting for other reasons.

EDHOC is adapted to primitives and protocols designed for the Internet of Things: EDHOC is built on CBOR and COSE which enables small message overhead and efficient parsing in constrained devices. EDHOC is not bound to a particular transport layer, but it is recommended to transport the EDHOC message in CoAP payloads. EDHOC is not bound to a particular communication security protocol but works off-the-shelf with OSCORE [RFC8613] providing the necessary input parameters with required properties. Maximum code complexity (ROM/Flash) is often a constraint in many devices and by reusing already existing libraries, the additional code footprint for EDHOC + OSCORE can be kept very low.

1.2. Terminology and Requirements Language

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

The word "encryption" without qualification always refers to authenticated encryption, in practice implemented with an Authenticated Encryption with Additional Data (AEAD) algorithm, see [RFC5116].

Readers are expected to be familiar with the terms and concepts described in CBOR [I-D.ietf-cbor-7049bis], COSE [RFC8152], and CDDL [RFC8610]. The Concise Data Definition Language (CDDL) is used to express CBOR data structures [I-D.ietf-cbor-7049bis]. Examples of CBOR and CDDL are provided in Appendix A.1.

2. Background

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and (D)TLS 1.3 [RFC8446], EDHOC is built on a variant of the SIGMA protocol which provide identity protection of the initiator (SIGMA-I), and like (D)TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an authenticated encryption algorithm is shown in Figure 2.

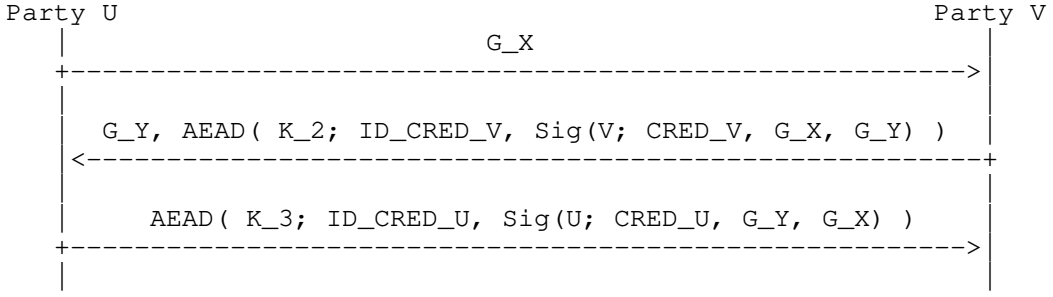


Figure 2: Authenticated encryption variant of the SIGMA-I protocol.

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive symmetric application keys.

- G_X and G_Y are the ECDH ephemeral public keys of U and V, respectively.
- CRED_U and CRED_V are the credentials containing the public authentication keys of U and V, respectively.
- ID_CRED_U and ID_CRED_V are data enabling the recipient party to retrieve the credential of U and V, respectively.
- Sig(U; .) and S(V; .) denote signatures made with the private authentication key of U and V, respectively.
- AEAD(K; .) denotes authenticated encryption with additional data using the key K derived from the shared secret. The authenticated

encryption MUST NOT be replaced by plain encryption, see Section 8.

In order to create a "full-fledged" protocol some additional protocol elements are needed. EDHOC adds:

- Explicit connection identifiers C_U, C_V chosen by U and V, respectively, enabling the recipient to find the protocol state.
- Transcript hashes TH_2, TH_3, TH_4 used for key derivation and as additional authenticated data.
- Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.
- Verification of a common preferred cipher suite (AEAD algorithm, ECDH algorithm, ECDH curve, signature algorithm):
  - U lists supported cipher suites in order of preference
  - V verifies that the selected cipher suite is the first supported cipher suite
- Method types and error handling.
- Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR, COSE, and CoAP libraries.

To simplify for implementors, the use of CBOR in EDHOC is summarized in Appendix A and test vectors including CBOR diagnostic notation are given in Appendix C.

3. EDHOC Overview

EDHOC consists of three flights (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. EDHOC messages are CBOR Sequences [I-D.ietf-cbor-sequence], where the first data item of message_1 is an int (TYPE) specifying the method (asymmetric, symmetric) and the correlation properties of the transport used.

While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0 structures, only a subset of the parameters is included in the EDHOC messages. After creating EDHOC message_3, Party U can derive symmetric application keys, and application protected data can therefore be sent in parallel with EDHOC message_3. The application may protect data using the algorithms (AEAD, HMAC, etc.) in the selected cipher suite and the connection identifiers (C_U, C_V). EDHOC may be used with the media type application/edhoc defined in Section 9.

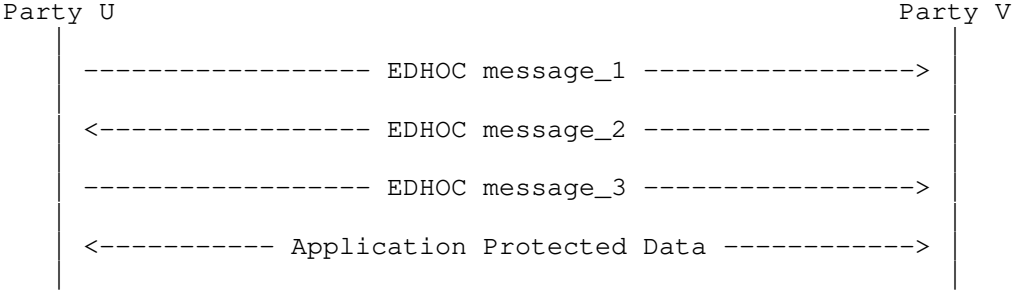


Figure 3: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or public key certificates. EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). When a public key infrastructure is used, the identity is included in the certificate and bound to the authentication key by trust in the certification authority. When the credential is manually distributed (PSK, RPK, self-signed certificate), the identity and authentication key is distributed out-of-band and bound together by trust in the distribution method. EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed, and that session keys are derived from the ECDH shared secret and the PSK.

EDHOC allows opaque application data (UAD and PAD) to be sent in the EDHOC messages. Unprotected Application Data (UAD_1, UAD_2) may be sent in message_1 and message_2 and can be e.g. be used to transfer access tokens that are protected outside of EDHOC. Protected application data (PAD_3) may be used to transfer any application data in message_3.

Cryptographically, EDHOC does not put requirements on the lower layers. EDHOC is not bound to a particular transport layer, and can be used in environments without IP. It is recommended to transport

the EDHOC message in CoAP payloads, see Section 7. An implementation may support only Party U or only Party V.

3.1. Cipher Suites

EDHOC cipher suites consist of an ordered set of COSE algorithms: an AEAD algorithm, an HMAC algorithm, an ECDH curve, a signature algorithm, and signature algorithm parameters. The signature algorithm is not used when EDHOC is authenticated with symmetric keys. Each cipher suite is either identified with a pre-defined int label or with an array of labels and values from the COSE Algorithms and Elliptic Curves registries.

```
suite = int / [ 4*4 algs: int / tstr, ? para: any ]
```

This document specifies two pre-defined cipher suites.

0. [10, 5, 4, -8, 6]
   (AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA, Ed25519)

1. [10, 5, 1, -7, 1]
   (AES-CCM-16-64-128, HMAC 256/256, P-256, ES256, P-256)

3.2. Ephemeral Public Keys

The ECDH ephemeral public keys are formatted as a COSE_Key of type EC2 or OKP according to Sections 13.1 and 13.2 of [RFC8152], but only the x-coordinate is included in the EDHOC messages. For Elliptic Curve Keys of type EC2, compact representation as per [RFC6090] MAY be used also in the COSE_Key. If the COSE implementation requires an y-coordinate, any of the possible values of the y-coordinate can be used, see Appendix C of [RFC6090]. COSE [RFC8152] always use compact output for Elliptic Curve Keys of type EC2.

3.3. Key Derivation

Key and IV derivation SHALL be performed with HKDF [RFC5869] following the specification in Section 11 of [RFC8152] using the HMAC algorithm in the selected cipher suite. The pseudorandom key (PRK) is derived using HKDF-Extract [RFC5869]

```
PRK = HKDF-Extract( salt, IKM )
```

with the following input:

- The salt SHALL be the PSK when EDHOC is authenticated with symmetric keys, and the empty byte string when EDHOC is authenticated with asymmetric keys. The PSK is used as 'salt' to

simplify implementation. Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros (see Section 2.2 of [RFC5869]). For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string.

- The input keying material (IKM) SHALL be the ECDH shared secret G_XY as defined in Section 12.4.1 of [RFC8152]. When using the curve25519, the ECDH shared secret is the output of the X25519 function [RFC7748].

Example: Assuming use of HMAC 256/256 the extract phase of HKDF produces a PRK as follows:

```
PRK = HMAC-SHA-256( salt, G_XY )
```

where salt = 0x (the empty byte string) in the asymmetric case and salt = PSK in the symmetric case.

The keys and IVs used in EDHOC are derived from PRK using HKDF-Expand [RFC5869]

```
OKM = HKDF-Expand( PRK, info, L )
```

where L is the length of output keying material (OKM) in bytes and info is the CBOR encoding of a COSE_KDF_Context

```
info = [
  AlgorithmID,
  [ null, null, null ],
  [ null, null, null ],
  [ keyDataLength, h'', other ]
]
```

where

- AlgorithmID is an int or tstr, see below
- keyDataLength is a uint set to the length of output keying material in bits, see below
- other is a bstr set to one of the transcript hashes TH_2, TH_3, or TH_4 as defined in Sections 4.3.1, 4.4.1, and 3.3.1.

For message_2 and message_3, the keys K_2 and K_3 SHALL be derived using transcript hashes TH_2 and TH_3 respectively. The key SHALL be derived using AlgorithmID set to the integer value of the AEAD in the

selected cipher suite, and keyDataLength equal to the key length of the AEAD.

If the AEAD algorithm uses an IV, then IV_2 and IV_3 for message_2 and message_3 SHALL be derived using the transcript hashes TH_2 and TH_3 respectively. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in Section 12.1.2. of [RFC8152], and keyDataLength equal to the IV length of the AEAD.

Assuming the output OKM length L is smaller than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation

```
OKM = first L bytes of HMAC( PRK, info || 0x01 )
```

where || means byte string concatenation.

Example: Assuming use of the algorithm AES-CCM-16-64-128 and HMAC 256/256, K_i and IV_i are therefore the first 16 and 13 bytes, respectively, of

```
HMAC-SHA-256( PRK, info || 0x01 )
```

calculated with (AlgorithmID, keyDataLength) = (10, 128) and (AlgorithmID, keyDataLength) = ("IV-GENERATION", 104), respectively.

3.3.1. EDHOC-Exporter Interface

Application keys and other application specific data can be derived using the EDHOC-Exporter interface defined as:

```
EDHOC-Exporter( label, length ) = HKDF-Expand( PRK, info, length )
```

The output of the EDHOC-Exporter function SHALL be derived using AlgorithmID = label, keyDataLength = 8 * length, and other = TH_4 where label is a tstr defined by the application and length is a uint defined by the application. The label SHALL be different for each different exporter value. The transcript hash TH_4 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

```
TH_4 = H( TH_3, CIPHERTEXT_3 )
```

where H() is the hash function in the HMAC algorithm. Example use of the EDHOC-Exporter is given in Sections 3.3.2 and 7.1.1.

3.3.2. EDHOC PSK Chaining

An application using EDHOC may want to derive new PSKs to use for authentication in future EDHOC exchanges. In this case, the new PSK and the ID_PSK 'kid_value' parameter SHOULD be derived as follows where length is the key length (in bytes) of the AEAD Algorithm.

```
PSK    = EDHOC-Exporter( "EDHOC Chaining PSK", length )
ID_PSK = EDHOC-Exporter( "EDHOC Chaining ID_PSK", 4 )
```

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

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

- Only Party V SHALL have access to the private authentication key of Party V,
- Only Party U SHALL have access to the private authentication key of Party U,
- Party U is able to retrieve Party V's public authentication key using ID_CRED_V,
- Party V is able to retrieve Party U's public authentication key using ID_CRED_U,

where the identifiers ID_CRED_U and ID_CRED_V are COSE header_maps, i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]). ID_CRED_U and ID_CRED_V need to contain parameters that can identify a public authentication key, see Appendix A.2. In the following we give some examples of possible COSE header parameters.

Raw public keys are most optimally stored as COSE_Key objects and identified with a 'kid' parameter (see [RFC8152]):

- ID_CRED_x = { 4 : kid_value }, where kid_value : bstr, for x = U or V.

Public key certificates can be identified in different ways. Several header parameters for identifying X.509 certificates are defined in [I-D.ietf-cose-x509] (the exact labels are TBD):

- by a hash value with the 'x5t' parameter;
  - ID_CRED_x = { TBD1 : COSE_CertHash }, for x = U or V,

- by a URL with the 'x5u' parameter;
  - ID_CRED_x = { TBD2 : uri }, for x = U or V,
- or by a bag of certificates with the 'x5bag' parameter;
  - ID_CRED_x = { TBD3 : COSE_X509 }, for x = U or V.
- by a certificate chain with the 'x5chain' parameter;
  - ID_CRED_x = { TBD4 : COSE_X509 }, for x = U or V,

In the latter two examples, ID_CRED_U and ID_CRED_V contain the actual credential used for authentication. The purpose of ID_CRED_U and ID_CRED_V is to facilitate retrieval of a public authentication key and when they do not contain the actual credential, they may be very short. It is RECOMMENDED that they uniquely identify the public authentication key as the recipient may otherwise have to try several keys. ID_CRED_U and ID_CRED_V are transported in the ciphertext, see Section 4.3.2 and Section 4.4.2.

The actual credentials CRED_U and CRED_V (e.g. a COSE_Key or a single X.509 certificate) are signed by party U and V, respectively to prevent duplicate-signature key selection (DSKS) attacks, see Section 4.4.1 and Section 4.3.1. Party U and Party V MAY use different types of credentials, e.g. one uses RPK and the other uses certificate. When included in the signature payload, COSE_Keys of type OKP SHALL only include the parameters 1 (kty), -1 (crv), and -2 (x-coordinate). COSE_Keys of type EC2 SHALL only include the parameters 1 (kty), -1 (crv), -2 (x-coordinate), and -3 (y-coordinate). The parameters SHALL be encoded in decreasing order.

The connection identifiers C_U and C_V do not have any cryptographic purpose in EDHOC. They contain information facilitating retrieval of the protocol state and may therefore be very short. The connection identifier MAY be used with an application protocol (e.g. OSCORE) for which EDHOC establishes keys, in which case the connection identifiers SHALL adhere to the requirements for that protocol. Each party choses a connection identifier it desires the other party to use in outgoing messages.

The first data item of message_1 is an int TYPE = 4 * method + corr specifying the method and the correlation properties of the transport used. corr = 0 is used when there is no external correlation mechanism. corr = 1 is used when there is an external correlation mechanism (e.g. the Token in CoAP) that enables Party U to correlate message_1 and message_2. corr = 2 is used when there is an external correlation mechanism that enables Party V to correlate message_2 and

message_3. corr = 3 is used when there is an external correlation mechanism that enables the parties to correlate all the messages. The use of the correlation parameter is exemplified in Section 7.1.

1 byte connection and credential identifiers are realistic in many scenarios as most constrained devices only have a few keys and connections. In cases where a node only has one connection or key, the identifiers may even be the empty byte string.

EDHOC with asymmetric key authentication is illustrated in Figure 4.

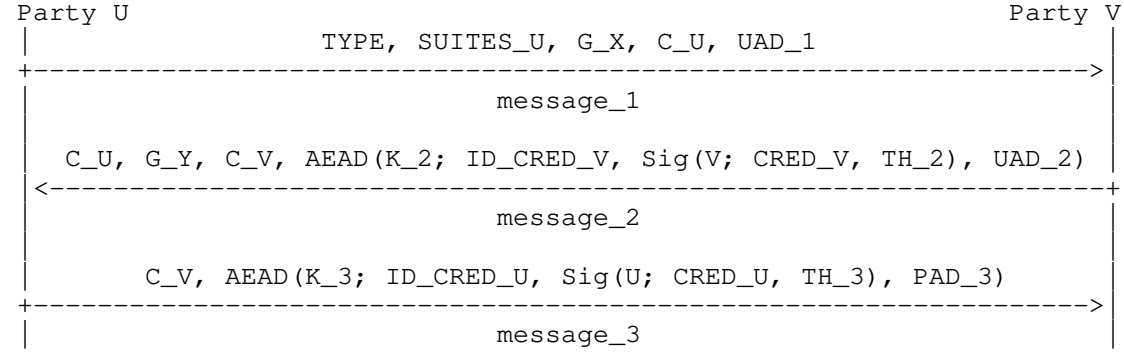


Figure 4: Overview of EDHOC with asymmetric key authentication.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
message_1 = (
  TYPE : int,
  SUITES_U : suite / [ index : uint, 2* suite ],
  G_X : bstr,
  C_U : bstr,
  ? UAD_1 : bstr,
)
```

where:

- TYPE = 4 * method + corr, where the method = 0 and the correlation parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- SUITES_U - cipher suites which Party U supports in order of decreasing preference. One cipher suite is selected. If a single cipher suite is conveyed then that cipher suite is selected. If multiple cipher suites are conveyed then zero-based index (i.e. 0 for the first suite, 1 for the second suite, etc.) identifies the selected cipher suite out of the array elements listing the cipher suites (see Section 6).

- G_X - the x-coordinate of the ephemeral public key of Party U

- C_U - variable length connection identifier

- UAD_1 - bstr containing unprotected opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- The supported cipher suites and the order of preference MUST NOT be changed based on previous error messages. However, the list SUITES_U sent to Party V MAY be truncated such that cipher suites which are the least preferred are omitted. The amount of truncation MAY be changed between sessions, e.g. based on previous error messages (see next bullet), but all cipher suites which are more preferred than the least preferred cipher suite in the list MUST be included in the list.

- Determine the cipher suite to use with Party V in message_1. If Party U previously received from Party V an error message to message_1 with diagnostic payload identifying a cipher suite that U supports, then U SHALL use that cipher suite. Otherwise the first cipher suite in SUITES_U MUST be used.

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. Let G_X be the x-coordinate of the ephemeral public key.

- Choose a connection identifier C_U and store it for the length of the protocol.

- Encode message_1 as a sequence of CBOR encoded data items as specified in Section 4.2.1

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Decode message_1 (see Appendix A.1).

- Verify that the selected cipher suite is supported and that no prior cipher suites in SUITES_U are supported.

- Validate that there is a solution to the curve definition for the given x-coordinate G_X.

- Pass UAD_1 to the application.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued. If V does not support the selected cipher suite, then SUITES_V MUST include one or more supported cipher suites. If V does not support the selected cipher suite, but supports another cipher suite in SUITES_U, then SUITES_V MUST include the first supported cipher suite in SUITES_U.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 and data_2 SHALL be CBOR Sequences (see Appendix A.1) as defined below

```
message_2 = (
  data_2,
  CIPHERTEXT_2 : bstr,
)

data_2 = (
  ? C_U : bstr,
  G_Y : bstr,
  C_V : bstr,
)
```

where:

- G_Y - the x-coordinate of the ephemeral public key of Party V

- C_V - variable length connection identifier

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- If TYPE mod 4 equals 1 or 3, C_U is omitted, otherwise C_U is not omitted.

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. Let G_Y be the x-coordinate of the ephemeral public key.

- Choose a connection identifier C_V and store it for the length of the protocol.

- Compute the transcript hash TH_2 = H(message_1, data_2) where H() is the hash function in the HMAC algorithm. The transcript hash TH_2 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

- Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party V, and the parameters below. Note that only 'signature' of the COSE_Sign1 object is used to create message_2, see next bullet. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_V

  * payload = CRED_V

  * external_aad = TH_2

  * ID_CRED_V - identifier to facilitate retrieval of CRED_V, see Section 4.1

  * CRED_V - bstr credential containing the credential of Party V, e.g. its public authentication key or X.509 certificate see Section 4.1. The public key must be a signature key. Note that if objects that are not bstr are used, such as COSE_Key for public authentication keys, these objects must be wrapped in a CBOR bstr.

  COSE constructs the input to the Signature Algorithm as follows:

  * The key is the private authentication key of V.

  * The message M to be signed is the CBOR encoding of:

    ["Signature1", << ID_CRED_V >>, TH_2, CRED_V]

- Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, IV_2, and the parameters below. Note that only 'ciphertext' of the COSE_Encrypt0 object is used to create message_2, see next bullet. The protected header SHALL be empty. The unprotected header (not

included in the EDHOC message) MAY contain parameters (e.g. 'alg').

* plaintext = (ID_CRED_V / kid_value, signature, ? UAD_2)

* external_aad = TH_2

* UAD_2 = bstr containing opaque unprotected application data

where signature is taken from the COSE_Sign1 object, ID_CRED_V is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]), and kid_value is a bstr. If ID_CRED_V contains a single 'kid' parameter, i.e., ID_CRED_V = { 4 : kid_value }, only kid_value is conveyed in the plaintext.

COSE constructs the input to the AEAD [RFC5116] as follows:

* Key K = K_2

* Nonce N = IV_2

* Plaintext P = (ID_CRED_V / kid_value, signature, ? UAD_2)

* Associated data A = ["Encrypt0", h'', TH_2]

- Encode message_2 as a sequence of CBOR encoded data items as specified in Section 4.3.1. CIPHERTEXT_2 is the COSE_Encrypt0 ciphertext.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Decode message_2 (see Appendix A.1).

- Retrieve the protocol state using the connection identifier C_U and/or other external information such as the CoAP Token and the 5-tuple.

- Validate that there is a solution to the curve definition for the given x-coordinate G_Y.

- Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, and IV_2.

- Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite and the public authentication key of Party V.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 and data_3 SHALL be CBOR Sequences (see Appendix A.1) as defined below

```
message_3 = (
  data_3,
  CIPHERTEXT_3 : bstr,
)

data_3 = (
  ? C_V : bstr,
)
```

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- If TYPE mod 4 equals 2 or 3, C_V is omitted, otherwise C_V is not omitted.

- Compute the transcript hash TH_3 = H(TH_2 , CIPHERTEXT_2, data_3) where H() is the hash function in the HMAC algorithm. The transcript hash TH_3 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

- Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party U, and the parameters below. Note that only 'signature' of the COSE_Sign1 object is used to create message_3, see next bullet. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_U

  * payload = CRED_U

  * external_aad = TH_3

* ID_CRED_U - identifier to facilitate retrieval of CRED_U, see Section 4.1

* CRED_U - bstr credential containing the credential of Party U, e.g. its public authentication key or X.509 certificate see Section 4.1. The public key must be a signature key. Note that if objects that are not bstr are used, such as COSE_Key for public authentication keys, these objects must be wrapped in a CBOR bstr.

COSE constructs the input to the Signature Algorithm as follows:

* The key is the private authentication key of U.

* The message M to be signed is the CBOR encoding of:

  ["Signature1", << ID_CRED_U >>, TH_3, CRED_U]

o Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, and IV_3 and the parameters below. Note that only 'ciphertext' of the COSE_Encrypt0 object is used to create message_3, see next bullet. The protected header SHALL be empty. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * plaintext = (ID_CRED_U / kid_value, signature, ? PAD_3)

  * external_aad = TH_3

  * PAD_3 = bstr containing opaque protected application data

  where signature is taken from the COSE_Sign1 object, ID_CRED_U is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]), and kid_value is a bstr. If ID_CRED_U contains a single 'kid' parameter, i.e., ID_CRED_U = { 4 : kid_value }, only kid_value is conveyed in the plaintext.

  COSE constructs the input to the AEAD [RFC5116] as follows:

  * Key K = K_3

  * Nonce N = IV_2

  * Plaintext P = (ID_CRED_U / kid_value, signature, ? PAD_3)

  * Associated data A = ["Encrypt0", h'', TH_3]

- Encode message_3 as a sequence of CBOR encoded data items as specified in Section 4.4.1. CIPHERTEXT_3 is the COSE_Encrypt0 ciphertext.

- Pass the connection identifiers (C_U, C_V) and the selected cipher suite to the application. The application can now derive application keys using the EDHOC-Exporter interface.

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Decode message_3 (see Appendix A.1).

- Retrieve the protocol state using the connection identifier C_V and/or other external information such as the CoAP Token and the 5-tuple.

- Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, and IV_3.

- Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite and the public authentication key of Party U.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.

- Pass PAD_3, the connection identifiers (C_U, C_V), and the selected cipher suite to the application. The application can now derive application keys using the EDHOC-Exporter interface.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

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

- Only Party U and Party V SHALL have access to the PSK,

- Party V is able to retrieve the PSK using ID_PSK.

where the identifier ID_PSK is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]) containing

COSE header parameter that can identify a pre-shared key. Pre-shared keys are typically stored as COSE_Key objects and identified with a 'kid' parameter (see [RFC8152]):

o ID_PSK = { 4 : kid_value } , where kid_value : bstr

The purpose of ID_PSK is to facilitate retrieval of the PSK and in the case a 'kid' parameter is used it may be very short. It is RECOMMENDED that it uniquely identify the PSK as the recipient may otherwise have to try several keys.

EDHOC with symmetric key authentication is illustrated in Figure 5.

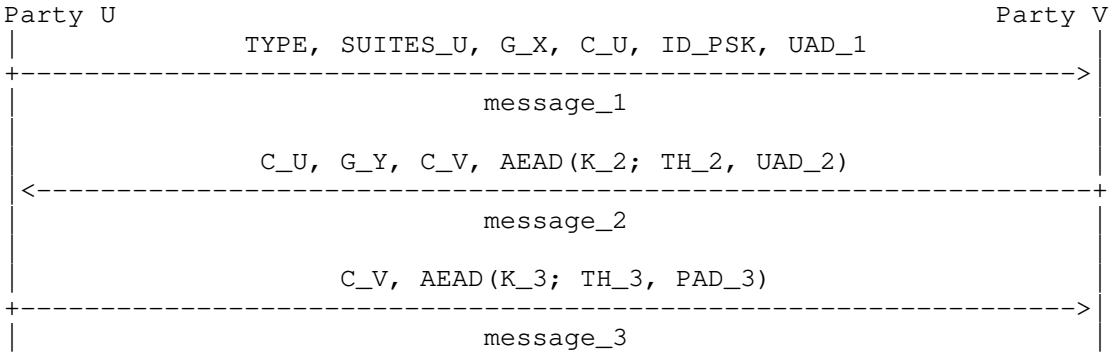


Figure 5: Overview of EDHOC with symmetric key authentication.

EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication. In the following subsections the differences compared to EDHOC with asymmetric key authentication are described.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
message_1 = (
  TYPE : int,
  SUITES_U : suite / [ index : uint, 2* suite ],
  G_X : bstr,
  C_U : bstr,
  ID_PSK : header_map // kid_value : bstr,
  ? UAD_1 : bstr,
)
```

where:

- TYPE = 4 * method + corr, where the method = 1 and the connection parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- ID_PSK - identifier to facilitate retrieval of the pre-shared key. If ID_PSK contains a single 'kid' parameter, i.e., ID_PSK = { 4 : kid_value }, with kid_value: bstr, only kid_value is conveyed.

5.3. EDHOC Message 2

5.3.1. Processing of Message 2

- COSE_Sign1 is not used.

- COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, IV_2, and the following parameters. The protected header SHALL be empty. The unprotected header MAY contain parameters (e.g. 'alg').

  * external_aad = TH_2

  * plaintext = ? UAD_2

  * UAD_2 = bstr containing opaque unprotected application data

5.4. EDHOC Message 3

5.4.1. Processing of Message 3

- COSE_Sign1 is not used.

- COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, IV_3, and the following parameters. The protected header SHALL be empty. The unprotected header MAY contain parameters (e.g. 'alg').

  * external_aad = TH_3

  * plaintext = ? PAD_3

  * PAD_3 = bstr containing opaque protected application data

6. Error Handling

6.1. EDHOC Error Message

This section defines a message format for the EDHOC error message, used during the protocol. An EDHOC error message can be sent by both parties as a reply to any non-error EDHOC message. After sending an error message, the protocol MUST be discontinued. Errors at the EDHOC layer are sent as normal successful messages in the lower layers (e.g. CoAP POST and 2.04 Changed). An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
error = (
  ? C_x : bstr,
  ERR_MSG : tstr,
  ? SUITES_V : suite / [ 2* suite ],
)
```

where:

- C_x - if error is sent by Party V and TYPE mod 4 equals 0 or 2 then C_x is set to C_U, else if error is sent by Party U and TYPE mod 4 equals 0 or 1 then C_x is set to C_V, else C_x is omitted.

- ERR_MSG - text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252]. ERR_MSG MAY be a 0-length text string.

- SUITES_V - cipher suites from SUITES_U or the EDHOC cipher suites registry that V supports. Note that SUITES_V only contains the values from the EDHOC cipher suites registry and no index. SUITES_V MUST only be included in replies to message_1.

6.1.1. Example Use of EDHOC Error Message with SUITES_V

Assuming that Party U supports the five cipher suites {5, 6, 7, 8, 9} in decreasing order of preference, Figures 6 and 7 show examples of how Party U can truncate SUITES_U and how SUITES_V is used by Party V to give Party U information about the cipher suites that Party V supports. In Figure 6, Party V supports cipher suite 6 but not the selected cipher suite 5.

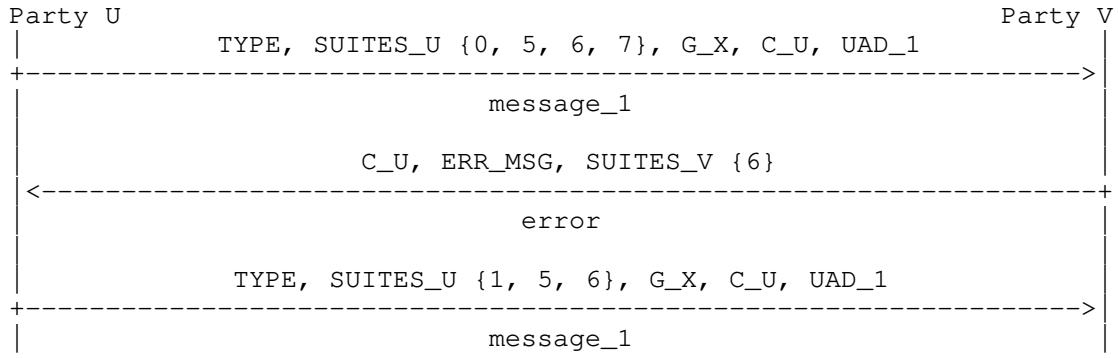


Figure 6: Example use of error message with SUITES_V.

In Figure 7, Party V supports cipher suite 7 but not cipher suites 5 and 6.

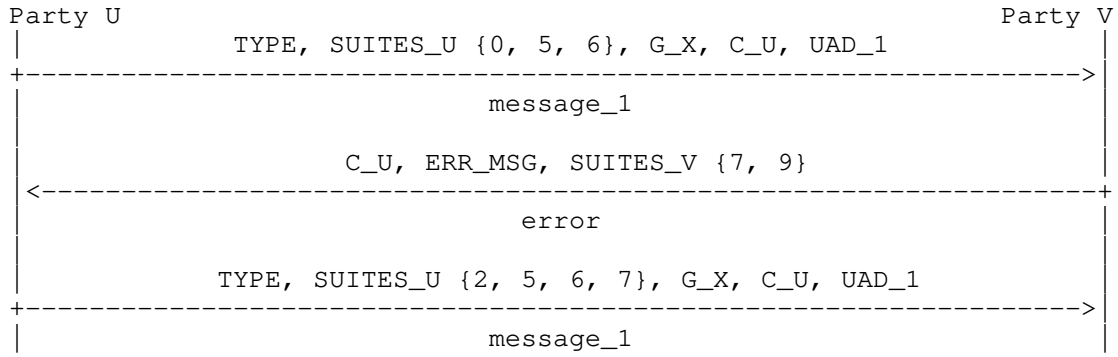


Figure 7: Example use of error message with SUITES_V.

As Party U's list of supported cipher suites and order of preference is fixed, and Party V only accepts message_1 if the selected cipher suite is the first cipher suite in SUITES_U that Party V supports, the parties can verify that the selected cipher suite is the most preferred (by Party U) cipher suite supported by both parties. If the selected cipher suite is not the first cipher suite in SUITES_U that Party V supports, Party V will discontinue the protocol.

7. Transferring EDHOC and Deriving Application Keys

7.1. Transferring EDHOC in CoAP

It is recommended to transport EDHOC as an exchange of CoAP [RFC7252] messages. CoAP is a reliable transport that can preserve packet ordering and handle message duplication. CoAP can also perform

fragmentation and protect against denial of service attacks. It is recommended to carry the EDHOC flights in Confirmable messages, especially if fragmentation is used.

By default, the CoAP client is Party U and the CoAP server is Party V, but the roles SHOULD be chosen to protect the most sensitive identity, see Section 8. By default, EDHOC is transferred in POST requests and 2.04 (Changed) responses to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discovered e.g. using resource directory [I-D.ietf-core-resource-directory].

By default, the message flow is as follows: EDHOC message_1 is sent in the payload of a POST request from the client to the server's resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 (Changed) response. EDHOC message_3 or the EDHOC error message is sent from the client to the server's resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 (Changed) response.

An example of a successful EDHOC exchange using CoAP is shown in Figure 8. In this case the CoAP Token enables Party U to correlate message_1 and message_2 so the correlation parameter corr = 1.

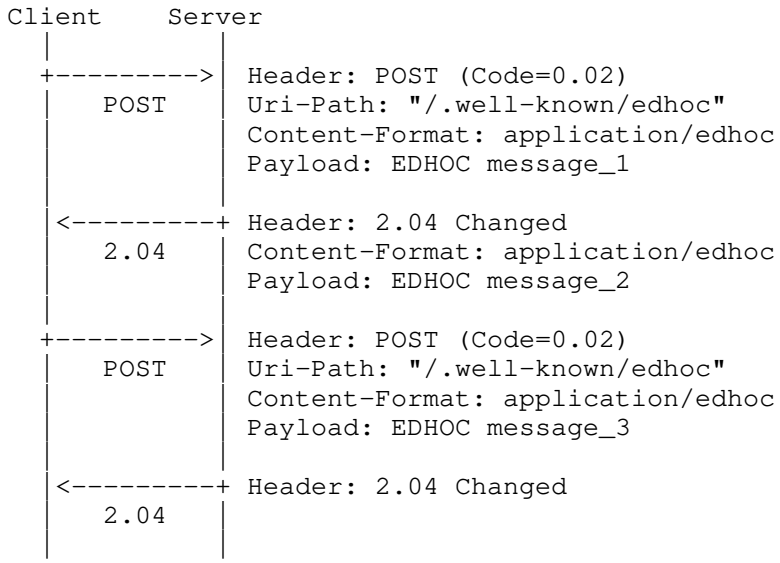


Figure 8: Transferring EDHOC in CoAP

The exchange in Figure 8 protects the client identity against active attackers and the server identity against passive attackers. An alternative exchange that protects the server identity against active attackers and the client identity against passive attackers is shown in Figure 9. In this case the CoAP Token enables Party V to correlate message_2 and message_3 so the correlation parameter corr = 2.

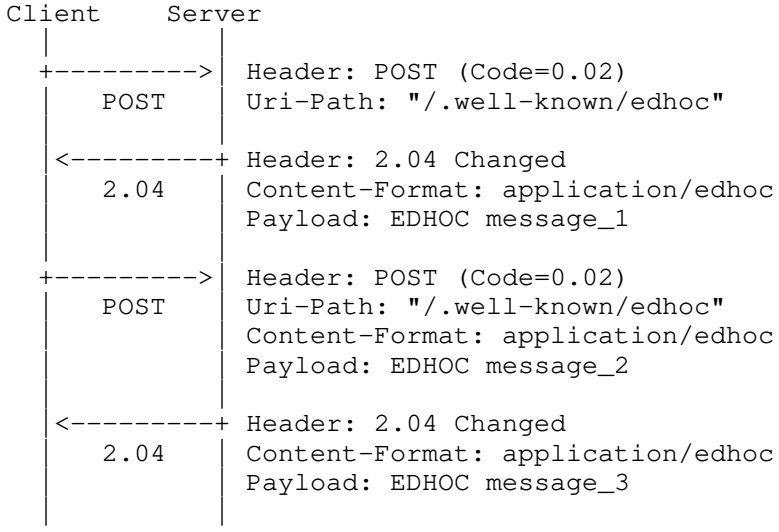


Figure 9: Transferring EDHOC in CoAP

To protect against denial-of-service attacks, the CoAP server MAY respond to the first POST request with a 4.01 (Unauthorized) containing an Echo option [I-D.ietf-core-echo-request-tag]. This forces the initiator to demonstrate its reachability at its apparent network address. If message fragmentation is needed, the EDHOC messages may be fragmented using the CoAP Block-Wise Transfer mechanism [RFC7959].

7.1.1. Deriving an OSCORE Context from EDHOC

When EDHOC is used to derive parameters for OSCORE [RFC8613], the parties must make sure that the EDHOC connection identifiers are unique, i.e. C_V MUST NOT be equal to C_U. The CoAP client and server MUST be able to retrieve the OSCORE protocol state using its chosen connection identifier and optionally other information such as the 5-tuple. In case that the CoAP client is party U and the CoAP server is party V:

- The client's OSCORE Sender ID is C_V and the server's OSCORE Sender ID is C_U, as defined in this document

- The AEAD Algorithm and the HMAC algorithms are the AEAD and HMAC algorithms in the selected cipher suite.

- The Master Secret and Master Salt are derived as follows where length is the key length (in bytes) of the AEAD Algorithm.

  ```
  Master Secret = EDHOC-Exporter( "OSCORE Master Secret", length )
  Master Salt   = EDHOC-Exporter( "OSCORE Master Salt", 8 )
  ```

7.2. Transferring EDHOC over Other Protocols

EDHOC may be transported over a different transport than CoAP. In this case the lower layers need to handle message loss, reordering, message duplication, fragmentation, and denial of service protection.

8. Security Considerations

8.1. Security Properties

EDHOC inherits its security properties from the theoretical SIGMA-I protocol [SIGMA]. Using the terminology from [SIGMA], EDHOC provides perfect forward secrecy, mutual authentication with aliveness, consistency, peer awareness, and identity protection. As described in [SIGMA], peer awareness is provided to Party V, but not to Party U. EDHOC also inherits Key Compromise Impersonation (KCI) resistance from SIGMA-I.

EDHOC with asymmetric authentication offers identity protection of Party U against active attacks and identity protection of Party V against passive attacks. The roles should be assigned to protect the most sensitive identity, typically that which is not possible to infer from routing information in the lower layers.

Compared to [SIGMA], EDHOC adds an explicit method type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. This protects against an attacker replaying messages or injecting messages from another session.

EDHOC also adds negotiation of connection identifiers and downgrade protected negotiation of cryptographic parameters, i.e. an attacker cannot affect the negotiated parameters. A single session of EDHOC does not include negotiation of cipher suites, but it enables Party V to verify that the selected cipher suite is the most preferred cipher suite by U which is supported by both U and V.

As required by [RFC7258], IETF protocols need to mitigate pervasive monitoring when possible. One way to mitigate pervasive monitoring is to use a key exchange that provides perfect forward secrecy. EDHOC therefore only supports methods with perfect forward secrecy. To limit the effect of breaches, it is important to limit the use of symmetrical group keys for bootstrapping. EDHOC therefore strives to make the additional cost of using raw public keys and self-signed certificates as small as possible. Raw public keys and self-signed certificates are not a replacement for a public key infrastructure, but SHOULD be used instead of symmetrical group keys for bootstrapping.

Compromise of the long-term keys (PSK or private authentication keys) does not compromise the security of completed EDHOC exchanges. Compromising the private authentication keys of one party lets the attacker impersonate that compromised party in EDHOC exchanges with other parties, but does not let the attacker impersonate other parties in EDHOC exchanges with the compromised party. Compromising the PSK lets the attacker impersonate Party U in EDHOC exchanges with Party V and impersonate Party V in EDHOC exchanges with Party U. Compromise of the HDKF input parameters (ECDH shared secret and/or PSK) leads to compromise of all session keys derived from that compromised shared secret. Compromise of one session key does not compromise other session keys.

8.2. Cryptographic Considerations

The security of the SIGMA protocol requires the MAC to be bound to the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism. EDHOC implements SIGMA-I using the same Sign-then-MAC approach as TLS 1.3.

To reduce message overhead EDHOC does not use explicit nonces and instead rely on the ephemeral public keys to provide randomness to each session. A good amount of randomness is important for the key generation, to provide liveness, and to protect against interleaving attacks. For this reason, the ephemeral keys MUST NOT be reused, and both parties SHALL generate fresh random ephemeral key pairs.

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

The data rates in many IoT deployments are very limited. Given that the application keys are protected as well as the long-term authentication keys they can often be used for years or even decades before the cryptographic limits are reached. If the application keys established through EDHOC need to be renewed, the communicating parties can derive application keys with other labels or run EDHOC again.

8.3. Cipher Suites

Cipher suite number 0 (AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519, Ed25519) is mandatory to implement. For many constrained IoT devices it is problematic to support more than one cipher suites, so some deployments with P-256 may not support the mandatory cipher suite. This is not a problem for local deployments.

The HMAC algorithm HMAC 256/64 (HMAC w/ SHA-256 truncated to 64 bits) SHALL NOT be supported for use in EDHOC.

8.4. Unprotected Data

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to UAD_1, ID_CRED_V, UAD_2, and ERR_MSG in the asymmetric case, and ID_PSK, UAD_1, and ERR_MSG in the symmetric case. Using the same ID_PSK or UAD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. The communicating parties may therefore anonymize ID_PSK. Another consideration is that the list of supported cipher suites may be used to identify the application.

Party U and V must also make sure that unauthenticated data does not trigger any harmful actions. In particular, this applies to UAD_1 and ERR_MSG in the asymmetric case, and ID_PSK, UAD_1, and ERR_MSG in the symmetric case.

8.5. Denial-of-Service

EDHOC itself does not provide countermeasures against Denial-of-Service attacks. By sending a number of new or replayed message_1 an attacker may cause Party V to allocate state, perform cryptographic operations, and amplify messages. To mitigate such attacks, an implementation SHOULD rely on lower layer mechanisms such as the Echo option in CoAP [I-D.ietf-core-echo-request-tag] that forces the initiator to demonstrate reachability at its apparent network address.

8.6. Implementation Considerations

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. As each pseudoranom number must only be used once, an implementation need to get a new truly random seed after reboot, or continously store state in nonvolatile memory, see ([RFC8613], Appendix B.1.1) for issues and solution approaches for writing to nonvolatile memory. If ECDSA is supported, "deterministic ECDSA" as specified in [RFC6979] is RECOMMENDED.

The referenced processing instructions in [SP-800-56A] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed. The ECDH shared secret, keys (K_2, K_3), and IVs (IV_2, IV_3) MUST be secret. Implementations should provide countermeasures to side-channel attacks such as timing attacks.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported. The private authentication keys and the PSK (even though it is used as salt) MUST be kept secret.

Party U and V are allowed to select the connection identifiers C_U and C_V, respectively, for the other party to use in the ongoing EDHOC protocol as well as in a subsequent application protocol (e.g. OSCORE [RFC8613]). The choice of connection identifier is not security critical in EDHOC but intended to simplify the retrieval of the right security context in combination with using short identifiers. If the wrong connection identifier of the other party is used in a protocol message it will result in the receiving party not being able to retrieve a security context (which will terminate the protocol) or retrieve the wrong security context (which also terminates the protocol as the message cannot be verified).

Party V MUST finish the verification step of message_3 before passing PAD_3 to the application.

If two nodes unintentionally initiate two simultaneous EDHOC message exchanges with each other even if they only want to complete a single EDHOC message exchange, they MAY terminate the exchange with the lexicographically smallest G_X. If the two G_X values are equal, the received message_1 MUST be discarded to mitigate reflection attacks. Note that in the case of two simultaneous EDHOC exchanges where the nodes only complete one and where the nodes have different preferred

cipher suites, an attacker can affect which of the two nodes' preferred cipher suites will be used by blocking the other exchange.

8.7. Other Documents Referencing EDHOC

EDHOC has been analyzed in several other documents. A formal verification of EDHOC was done in [SSR18], an analysis of EDHOC for certificate enrollment was done in [Kron18], the use of EDHOC in LoRaWAN is analyzed in [LoRa1] and [LoRa2], the use of EDHOC in IoT bootstrapping is analyzed in [Perez18], and the use of EDHOC in 6TiSCH is described in [I-D.ietf-6tisch-dtsecurity-zerotouch-join].

9. IANA Considerations

9.1. EDHOC Cipher Suites Registry

IANA has created a new registry titled "EDHOC Cipher Suites" under the new heading "EDHOC". The registration procedure is "Expert Review". The columns of the registry are Value, Array, Description, and Reference, where Value is an integer and the other columns are text strings. The initial contents of the registry are:

Value: 1
Array: [10, 5, 1, -7, 1]
Desc: AES-CCM-16-64-128, HMAC 256/256, P-256, ES256, P-256
Reference: [[this document]]

Value: 0
Array: [10, 5, 4, -8, 6]
Desc: AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA, Ed25519
Reference: [[this document]]

Value: -5
Array:
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -6
Array:
Desc: Reserved for Private Use
Reference: [[this document]]

9.2. EDHOC Method Type Registry

IANA has created a new registry titled "EDHOC Method Type" under the new heading "EDHOC". The registration procedure is "Expert Review". The columns of the registry are Value, Description, and Reference,

where Value is an integer and the other columns are text strings. The initial contents of the registry are:

Value	Specification	Reference
0	EDHOC Authenticated with Asymmetric Keys	[[this document]]
1	EDHOC Authenticated with Symmetric Keys	[[this document]]

9.3. The Well-Known URI Registry

IANA has added the well-known URI 'edhoc' to the Well-Known URIs registry.

- URI suffix: edhoc
- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

9.4. Media Types Registry

IANA has added the media type 'application/edhoc' to the Media Types registry.

- Type name: application
- Subtype name: edhoc
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See Section 7 of this document.
- Interoperability considerations: N/A
- Published specification: [[this document]] (this document)
- Applications that use this media type: To be identified
- Fragment identifier considerations: N/A

- Additional information:
  - Magic number(s): N/A
  - File extension(s): N/A
  - Macintosh file type code(s): N/A
- Person & email address to contact for further information: See "Authors' Addresses" section.
- Intended usage: COMMON
- Restrictions on usage: N/A
- Author: See "Authors' Addresses" section.
- Change Controller: IESG

9.5. CoAP Content-Formats Registry

IANA has added the media type 'application/edhoc' to the CoAP Content-Formats registry.

- Media Type: application/edhoc
- Encoding:
- ID: TBD42
- Reference: [[this document]]

9.6. Expert Review Instructions

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

Expert reviewers should take into consideration the following points:

- Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Expert needs to make sure the values of algorithms are taken from the right registry, when that's required. Expert should consider requesting an opinion on the correctness of registered parameters from relevant IETF working groups. Encodings that do not meet

these objective of clarity and completeness should not be registered.

- Experts should take into account the expected usage of fields when approving point assignment. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

- Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

10. References

10.1. Normative References

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-07 (work in progress), August 2019.

[I-D.ietf-cbor-sequence]
Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", draft-ietf-cbor-sequence-01 (work in progress), August 2019.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-05 (work in progress), May 2019.

[I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Headers for carrying and referencing X.509 certificates", draft-ietf-cose-x509-03 (work in progress), August 2019.

[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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[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>.

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

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[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>.

[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>.

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

[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>.

[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>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

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

[SP-800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 3, April 2018, <https://doi.org/10.6028/NIST.SP.800-56Ar3>.

10.2. Informative References

[CborMe] Bormann, C., "CBOR Playground", May 2018, <http://cbor.me/>.

[I-D.hartke-core-e2e-security-reqs] Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.

[I-D.ietf-6tisch-dtsecurity-zerotouch-join] Richardson, M., "6tisch Zero-Touch Secure Join protocol", draft-ietf-6tisch-dtsecurity-zerotouch-join-04 (work in progress), July 2019.

[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-24 (work in progress), March 2019.

[I-D.ietf-ace-oscore-profile] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-08 (work in progress), July 2019.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J. and F. Palombini, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-03 (work in progress), March 2019.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-32 (work in progress), July 2019.

[Kron18] Krontiris, A., "Evaluation of Certificate Enrollment over Application Layer Security", May 2018, <https://www.nada.kth.se/~ann/exjobb/alexandros_krontiris.pdf>.

[LoRa1] Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S., Fernandez, P., Santa, J., Hernandez-Ramos, J., and A. Skarmeta, "Enhancing LoRaWAN Security through a Lightweight and Authenticated Key Management Approach", June 2018, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6021899/pdf/sensors-18-01833.pdf>.

[LoRa2] Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S., Fernandez, P., Santa, J., Hernandez-Ramos, J., and A. Skarmeta, "Internet Access for LoRaWAN Devices Considering Security Issues", June 2018, <https://ants.inf.um.es/~josesanta/doc/GIoTS1.pdf>.

[OPTLS] Krawczyk, H. and H. Wee, "The OPTLS Protocol and TLS 1.3", October 2015, <https://eprint.iacr.org/2015/978.pdf>.

[Perez18] Perez, S., Garcia-Carrillo, D., Marin-Lopez, R., Hernandez-Ramos, J., Marin-Perez, R., and A. Skarmeta, "Architecture of security association establishment based on bootstrapping technologies for enabling critical IoT infrastructures", October 2018, <http://www.anastacia-h2020.eu/publications/Architecture_of_security_association_establishment_based_on_bootstrapping_technologies_for_enabling_critical_IoT_infrastructures.pdf>.

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

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[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>.

[SSR18] Bruni, A., Sahl Joergensen, T., Groenbech Petersen, T., and C. Schuermann, "Formal Verification of Ephemeral Diffie-Hellman Over COSE (EDHOC)", November 2018, <https://www.springerprofessional.de/en/formal-verification-of-ephemeral-diffie-hellman-over-cose-edhoc/16284348>.

Appendix A. Use of CBOR, CDDL and COSE in EDHOC

This Appendix is intended to simplify for implementors not familiar with CBOR [I-D.ietf-cbor-7049bis], CDDL [RFC8610], COSE [RFC8152], and HKDF [RFC5869].

A.1. CBOR and CDDL

The Concise Binary Object Representation (CBOR) [I-D.ietf-cbor-7049bis] is a data format designed for small code size and small message size. CBOR builds on the JSON data model but extends it by e.g. encoding binary data directly without base64 conversion. In addition to the binary CBOR encoding, CBOR also has a diagnostic notation that is readable and editable by humans. The Concise Data Definition Language (CDDL) [RFC8610] provides a way to express structures for protocol messages and APIs that use CBOR. [RFC8610] also extends the diagnostic notation.

CBOR data items are encoded to or decoded from byte strings using a type-length-value encoding scheme, where the three highest order bits of the initial byte contain information about the major type. CBOR supports several different types of data items, in addition to integers (int, uint), simple values (e.g. null), byte strings (bstr), and text strings (tstr), CBOR also supports arrays [] of data items, maps {} of pairs of data items, and sequences [I-D.ietf-cbor-sequence] of data items. Some examples are given below. For a complete specification and more examples, see [I-D.ietf-cbor-7049bis] and [RFC8610]. We recommend implementors to get used to CBOR by using the CBOR playground [CborMe].

Diagnostic	Encoded	Type
1	0x01	unsigned integer
24	0x1818	unsigned integer
-24	0x37	negative integer
-25	0x3818	negative integer
null	0xf6	simple value
h'12cd'	0x4212cd	byte string
'12cd'	0x4431326364	byte string
"12cd"	0x6431326364	text string
{ 4 : h'cd' }	0xa10441cd	map
<< 1, 2, null >>	0x430102f6	byte string
[1, 2, null]	0x830102f6	array
(1, 2, null)	0x0102f6	sequence
1, 2, null	0x0102f6	sequence

EDHOC messages are CBOR Sequences [I-D.ietf-cbor-sequence]. The message format specification uses the construct '.cbor' enabling conversion between different CDDL types matching different CBOR items with different encodings. Some examples are given below.

A type (e.g. an uint) may be wrapped in a byte string (bstr):

CDDL Type	Diagnostic	Encoded
uint	24	0x1818
bstr .cbor uint	<< 24 >>	0x421818

A.2. COSE

CBOR Object Signing and Encryption (COSE) [RFC8152] describes how to create and process signatures, message authentication codes, and encryption using CBOR. COSE builds on JOSE, but is adapted to allow more efficient processing in constrained devices. EDHOC makes use of COSE_Key, COSE_Encrypt0, COSE_Sign1, and COSE_KDF_Context objects.

Appendix B. EDHOC Authenticated withDiffie-Hellman Keys

The SIGMA protocol is mainly optimized for PKI and certificates. The OPTLS protocol [OPTLS] shows how authentication can be provided by a MAC computed from an ephemeral-static ECDH shared secret. Instead of signature authentication keys, U and V would have Diffie-Hellman authentication keys G_U and G_V, respectively. This type of authentication keys could easily be used with RPK and would provide significant reductions in message sizes as the 64 bytes signature would be replaced by an 8 bytes MAC.

EDHOC authenticated with asymmetric Diffie-Hellman keys should have similar security properties as EDHOC authenticated with asymmetric signature keys with a few differences:

- Repudiation: In EDHOC authenticated with asymmetric signature keys, Party U could theoretically prove that Party V performed a run of the protocol by presenting the private ephemeral key, and vice versa. Note that storing the private ephemeral keys violates the protocol requirements. With asymmetric Diffie-Hellman key authentication, both parties can always deny having participated in the protocol, this is similar to EDHOC with symmetric key authentication.

- Key compromise impersonation (KCI): In EDHOC authenticated with asymmetric signature keys, EDHOC provides KCI protection against an attacker having access to the long term key or the ephemeral secret key. In EDHOC authenticated with symmetric keys, EDHOC provides KCI protection against an attacker having access to the ephemeral secret key, but not against an attacker having access to the long-term PSK. With asymmetric Diffie-Hellman key authentication, KCI protection would be provided against an attacker having access to the long-term Diffie-Hellman key, but not to an attacker having access to the ephemeral secret key. Note that the term KCI has typically been used for compromise of long-term keys, and that an attacker with access to the ephemeral secret key can only attack that specific protocol run.

TODO: Initial suggestion for key derivation, message formats, and processing

Appendix C. Test Vectors

This appendix provides detailed test vectors to ease implementation and ensure interoperability. In addition to hexadecimal, all CBOR data items and sequences are given in CBOR diagnostic notation. The test vectors use 1 byte key identifiers, 1 byte connection IDs, and the default mapping to CoAP where Party U is CoAP client (this means that corr = 1).

C.1. Test Vectors for EDHOC Authenticated with Asymmetric Keys (RPK)

Asymmetric EDHOC is used:

method (Asymmetric Authentication)
0

CoAP is used as transport:

corr (Party U is CoAP client)
1

No unprotected opaque application data is sent in the message exchanges.

The pre-defined Cipher Suite 0 is in place both on Party U and Party V, see Section 3.1.

C.1.1. Input for Party U

The following are the parameters that are set in Party U before the first message exchange.

Party U's private authentication key (32 bytes)
53 21 fc 01 c2 98 20 06 3a 72 50 8f c6 39 25 1d c8 30 e2 f7 68 3e b8 e3 8a f1 64 a5 b9 af 9b e3

Party U's public authentication key (32 bytes)
42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

kid value to identify U's public authentication key (1 bytes)
a2

This test vector uses COSE_Key objects to store the raw public keys. Moreover, EC2 keys with curve Ed25519 are used. That is in agreement with the Cipher Suite 0.

```
CRED_U =
<< {
  1:  1,
 -1:  6,
 -2:  h'424c756ab77cc6fdecf0b3ecfcffb75310c015bf5cba2ec0a236e6650c8ab9c7'
} >>
```

CRED_U (COSE_Key) (CBOR-encoded) (42 bytes)
58 28 a3 01 01 20 06 21 58 20 42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

Because COSE_Keys are used, and because kid = h'a2':

```
ID_CRED_U =
{
  4:  h'a2'
}
```

Note that since the map for ID_CRED_U contains a single 'kid' parameter, ID_CRED_U is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 4.4.2):

ID_CRED_U (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a2

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a2

C.1.2. Input for Party V

The following are the parameters that are set in Party V before the first message exchange.

Party V's private authentication key (32 bytes)
74 56 b3 a3 e5 8d 8d 26 dd 36 bc 75 d5 5b 88 63 a8 5d 34 72 f4 a0 1f 02 24 62 1b 1c b8 16 6d a9

Party V's public authentication key (32 bytes)
1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2 0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3

kid value to identify U's public authentication key (1 bytes)
a3

This test vector uses COSE_Key objects to store the raw public keys. Moreover, EC2 keys with curve Ed25519 are used. That is in agreement with the Cipher Suite 0.

```
CRED_V =
<< {
  1:  1,
 -1:  6,
 -2:  h'1b661ee5d5ef1672a2d877cd5bc20f4630dc78a114de659c7e504d0f529a6bd3'
} >>
```

CRED_V (COSE_Key) (CBOR-encoded) (42 bytes)
58 28 a3 01 01 20 06 21 58 20 1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2 0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3

Because COSE_Keys are used, and because kid = h'a3':

```
ID_CRED_V =
{
  4:  h'a3'
}
```

Note that since the map for ID_CRED_U contains a single 'kid' parameter, ID_CRED_U is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 4.4.2):

ID_CRED_V (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a3

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a3

C.1.3. Message 1

From the input parameters (in Appendix C.1.1):

TYPE (4 * method + corr)
1

suite
0

SUITES_U : suite
0

G_X (X-coordinate of the ephemeral public key of Party U) (32 bytes)
b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71 91 2d 6d b8 f4 af 98 0d 2d b8 3a

C_U (Connection identifier chosen by U) (1 bytes)
c3

No UAD_1 is provided, so UAD_1 is absent from message_1.

Message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  1,
  0,
  h'b1a3e89460e88d3a8d54211dc95f0b903ff205eb71912d6db8f4af980d2db83a',
  h'c3'
)
```

message_1 (CBOR Sequence) (38 bytes)
01 00 58 20 b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71 91 2d 6d b8 f4 af 98 0d 2d b8 3a 41 c3

C.1.4. Message 2

Since TYPE mod 4 equals 1, C_U is omitted from data_2.

G_Y (X-coordinate of the ephemeral public key of Party V) (32 bytes)
8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c 32 0e
5d 49 f3 02 a9 64 74

C_V (Connection identifier chosen by V) (1 bytes)
c4

Data_2 is constructed, as the CBOR Sequence of the CBOR data items above.

```
data_2 =
(
  h'8db577f9b9c2744798987db557bf31ca48acd205a9db8c320e5d49f302a96474',
  h'c4'
)
```

data_2 (CBOR Sequence) (36 bytes)
58 20 8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c
32 0e 5d 49 f3 02 a9 64 74 41 c4

From data_2 and message_1 (from Appendix C.1.3), compute the input to the transcript hash TH_2 = H(message_1, data_2), as a CBOR Sequence of these 2 data items.

(message_1, data_2) (CBOR Sequence)
(74 bytes)
01 00 58 20 b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71
91 2d 6d b8 f4 af 98 0d 2d b8 3a 41 c3 58 20 8d b5 77 f9 b9 c2 74 47 98 98
7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c 32 0e 5d 49 f3 02 a9 64 74 41 c4

And from there, compute the transcript hash TH_2 = SHA-256(message_1, data_2)

TH_2 value (32 bytes)
55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68
1d c2 af dd 87 03 55

When encoded as a CBOR bstr, that gives:

TH_2 (CBOR-encoded) (34 bytes)
58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11
da 68 1d c2 af dd 87 03 55

C.1.4.1. Signature Computation

COSE_Sign1 is computed with the following parameters. From Appendix C.1.2:

- protected = bstr .cbor ID_CRED_V
- payload = CRED_V

And from Appendix C.1.4:

- external_aad = TH_2

The Sig_structure M_V to be signed is: ["Signature1", << ID_CRED_V >>, TH_2, CRED_V] , as defined in Section 4.3.2:

```
M_V =
[
  "Signature1",
  << { 4: h'a3' } >>,
  h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd870355',
  << {
    1:  1,
   -1:  6,
   -2:  h'1b661ee5d5ef1672a2d877cd5bc20f4630dc78a114de659c7e504d0f529a6b
          d3'
  } >>
]
```

Which encodes to the following byte string ToBeSigned:

```
M_V (message to be signed with Ed25519) (CBOR-encoded) (93 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 44 a1 04 41 a3 58 20 55 50 b3 dc 59 84
b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03
55 58 28 a3 01 01 20 06 21 58 20 1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2
0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3
```

The message is signed using the private authentication key of V, and produces the following signature:

```
V's signature (64 bytes)
52 3d 99 6d fd 9e 2f 77 c7 68 71 8a 30 c3 48 77 8c 5e b8 64 dd 53 7e 55 5e
4a 00 05 e2 09 53 07 13 ca 14 62 0d e8 18 7e 81 99 6e e8 04 d1 53 b8 a1 f6
08 49 6f dc d9 3d 30 fc 1c 8b 45 be cc 06
```

C.1.4.2. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the asymmetric case, salt is the empty byte string.

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
c6 1e 09 09 a1 9d 64 24 01 63 ec 26 2e 9c c4 f8 8c e7 7b e1 23 c5 ab 53 8d
26 b0 69 22 a5 20 67
```

From there, PRK is computed:

```
PRK (32 bytes)
ba 9c 2c a1 c5 62 14 a6 e0 f6 13 ed a8 91 86 8a 4c a3 e3 fa bc c7 79 8f dc
01 60 80 07 59 16 71
```

Key K_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_2
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd
                870355' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_2) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 55 50 b3 dc 59 84 b0 20 9a
e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03 55
```

L is the length of K_2, so 16 bytes.

From these parameters, K_2 is computed:

K_2 (16 bytes)
da d7 44 af 07 c4 da 27 d1 f0 a3 8a 0c 4b 87 38

Nonce IV_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_2
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd
                870355' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_2) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33
2b 11 da 68 1d c2 af dd 87 03 55
```

L is the length of IV_2, so 13 bytes.

From these parameters, IV_2 is computed:

IV_2 (13 bytes)
fb a1 65 d9 08 da a7 8e 4f 84 41 42 d0

C.1.4.3. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that UAD_2 is omitted.

- empty protected header
- external_aad = TH_2
- plaintext = CBOR Sequence of the items kid_value, signature, in this order.

with kid_value taken from Appendix C.1.2, and signature as calculated in Appendix C.1.4.1.

The plaintext is the following:

```
P_2 (68 bytes)
41 a3 58 40 52 3d 99 6d fd 9e 2f 77 c7 68 71 8a 30 c3 48 77 8c 5e b8 64 dd
53 7e 55 5e 4a 00 05 e2 09 53 07 13 ca 14 62 0d e8 18 7e 81 99 6e e8 04 d1
53 b8 a1 f6 08 49 6f dc d9 3d 30 fc 1c 8b 45 be cc 06
```

From the parameters above, the Enc_structure A_2 is computed.

```
A_2 =
[
  "Encrypt0",
  h'',
  h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd870355'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2
6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03 55
```

The key and nonce used are defined in Appendix C.1.4.2:

- key = K_2

- nonce = IV_2

Using the parameters above, the ciphertext CIPHERTEXT_2 can be computed:

```
CIPHERTEXT_2 (76 bytes)
1e 6b fe 0e 77 99 ce f0 66 a3 4f 08 ef aa 90 00 6d b4 4c 90 1c f7 9b 23 85
3a b9 7f d8 db c8 53 39 d5 ed 80 87 78 3c f7 a4 a7 e0 ea 38 c2 21 78 9f a3
71 be 64 e9 3c 43 a7 db 47 d1 e3 fb 14 78 8e 96 7f dd 78 d8 80 78 e4 9b 78
bf
```

C.1.4.4. message_2

From the parameter computed in Appendix C.1.4 and Appendix C.1.4.3, message_2 is computed, as the CBOR Sequence of the following items: (G_Y, C_V, CIPHERTEXT_2).

```
message_2 =
(
  h'8db577f9b9c2744798987db557bf31ca48acd205a9db8c320e5d49f302a96474',
  h'c4',
  h'1e6bfe0e7799cef066a34f08efaa90006db44c901cf79b23853ab97fd8dbc85339d5ed
  8087783cf7a4a7e0ea38c221789fa371be64e93c43a7db47d1e3fb14788e967fdd78d880
  78e49b78bf'
)
```

Which encodes to the following byte string:

```
message_2 (CBOR Sequence) (114 bytes)
58 20 8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c
32 0e 5d 49 f3 02 a9 64 74 41 c4 58 4c 1e 6b fe 0e 77 99 ce f0 66 a3 4f 08
ef aa 90 00 6d b4 4c 90 1c f7 9b 23 85 3a b9 7f d8 db c8 53 39 d5 ed 80 87
78 3c f7 a4 a7 e0 ea 38 c2 21 78 9f a3 71 be 64 e9 3c 43 a7 db 47 d1 e3 fb
14 78 8e 96 7f dd 78 d8 80 78 e4 9b 78 bf
```

C.1.5. Message 3

Since TYPE mod 4 equals 1, C_V is not omitted from data_3.

```
C_V (1 bytes)
c4
```

Data_3 is constructed, as the CBOR Sequence of the CBOR data item above.

```
data_3 =
(
  h'c4'
)

data_3 (CBOR Sequence) (2 bytes)
41 c4
```

From data_3, CIPHERTEXT_2 (Appendix C.1.4.3), and TH_2 (Appendix C.1.4), compute the input to the transcript hash TH_2 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

```
( TH_2, CIPHERTEXT_2, data_3 )
(CBOR Sequence) (114 bytes)
58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11
da 68 1d c2 af dd 87 03 55 58 4c 1e 6b fe 0e 77 99 ce f0 66 a3 4f 08 ef aa
90 00 6d b4 4c 90 1c f7 9b 23 85 3a b9 7f d8 db c8 53 39 d5 ed 80 87 78 3c
f7 a4 a7 e0 ea 38 c2 21 78 9f a3 71 be 64 e9 3c 43 a7 db 47 d1 e3 fb 14 78
8e 96 7f dd 78 d8 80 78 e4 9b 78 bf 41 c4
```

And from there, compute the transcript hash TH_3 = SHA-256(TH_2 , CIPHERTEXT_2, data_3)

```
TH_3 value (32 bytes)
21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07
f3 e7 85 43 67 fc 22
```

When encoded as a CBOR bstr, that gives:

```
TH_3 (CBOR-encoded) (34 bytes)
58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a
79 07 f3 e7 85 43 67 fc 22
```

C.1.5.1. Signature Computation

COSE_Sign1 is computed with the following parameters. From Appendix C.1.2:

- protected = bstr .cbor ID_CRED_U
- payload = CRED_U

And from Appendix C.1.4:

- external_aad = TH_3

The Sig_structure M_V to be signed is: ["Signature1", << ID_CRED_U >>, TH_3, CRED_U] , as defined in Section 4.4.2:

```
M_U =
[
  "Signature1",
  << { 4: h'a2' } >>,
  h'734bef323d867a12956127c2e62ade42c0f119e5487750c0c31fd093376dceed',
  << {
    1:  1,
   -1:  6,
   -2:  h'424c756ab77cc6fdecf0b3ecfcffb75310c015bf5cba2ec0a236e6650c8ab9
   c7'
  } >>
]
```

Which encodes to the following byte string ToBeSigned:

M_U (message to be signed with Ed25519) (CBOR-encoded) (93 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 44 a1 04 41 a2 58 20 73 4b ef 32 3d 86
7a 12 95 61 27 c2 e6 2a de 42 c0 f1 19 e5 48 77 50 c0 c3 1f d0 93 37 6d ce
ed 58 28 a3 01 01 20 06 21 58 20 42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff
b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

The message is signed using the private authentication key of U, and produces the following signature:

U's signature (64 bytes)
5c 7d 7d 64 c9 61 c5 f5 2d cf 33 91 25 92 a1 af f0 2c 33 62 b0 e7 55 0e 4b
c5 66 b7 0c 20 61 f3 c5 f6 49 e5 ed 32 3d 30 a2 6c 61 2f bb 5c bd 25 f3 1c
27 22 8c ea ec 64 29 31 95 41 fe 07 8e 0e

C.1.5.2. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the asymmetric case, salt is the empty byte string.

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

G_XY (32 bytes)
c6 1e 09 09 a1 9d 64 24 01 63 ec 26 2e 9c c4 f8 8c e7 7b e1 23 c5 ab 53 8d
26 b0 69 22 a5 20 67

From there, PRK is computed:

PRK (32 bytes)
ba 9c 2c a1 c5 62 14 a6 e0 f6 13 ed a8 91 86 8a 4c a3 e3 fa bc c7 79 8f dc
01 60 80 07 59 16 71

Key K_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_3
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e78543
  67fc22' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_3) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 21 cc b6 78 b7 91 14 96 09
55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07 f3 e7 85 43 67 fc 22
```

L is the length of K_3, so 16 bytes.

From these parameters, K_3 is computed:

```
K_3 (16 bytes)
e1 ac d4 76 f5 96 a4 60 72 44 a8 da 8c ff 49 df
```

Nonce IV_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_3
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e78543
  67fc22' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_3) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e
37 4a 79 07 f3 e7 85 43 67 fc 22
```

L is the length of IV_3, so 13 bytes.

From these parameters, IV_3 is computed:

```
IV_3 (13 bytes)
de 53 02 13 ab a2 6a 47 1a 51 f3 d6 fb
```

C.1.5.3. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that PAD_3 is omitted.

- empty protected header
- external_aad = TH_3
- plaintext = CBOR Sequence of the items kid_value, signature, in this order.

with kid_value taken from Appendix C.1.1, and signature as calculated in Appendix C.1.5.1.

The plaintext is the following:

```
P_3 (68 bytes)
41 a2 58 40 5c 7d 7d 64 c9 61 c5 f5 2d cf 33 91 25 92 a1 af f0 2c 33 62 b0
e7 55 0e 4b c5 66 b7 0c 20 61 f3 c5 f6 49 e5 ed 32 3d 30 a2 6c 61 2f bb 5c
bd 25 f3 1c 27 22 8c ea ec 64 29 31 95 41 fe 07 8e 0e
```

From the parameters above, the Enc_structure A_3 is computed.

```
A_3 =
[
  "Encrypt0",
  h'',
  h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e7854367fc22'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b
90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07 f3 e7 85 43 67 fc 22
```

The key and nonce used are defined in Appendix C.1.4.2:

- key = K_3
- nonce = IV_3

Using the parameters above, the ciphertext CIPHERTEXT_3 can be computed:

CIPHERTEXT_3 (76 bytes)
```
de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87 d9 3a f8 35 57 9c 2d bf 1b 9e 2f
b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3 5d 51 5b 4d 7d 64 83 f5 09 61 43
b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57 83 1d d2 e5 bd 04 04 38 60 14 0d
c8
```

C.1.5.4. message_3

From the parameter computed in Appendix C.1.5 and Appendix C.1.5.3, message_3 is computed, as the CBOR Sequence of the following items: (C_V, CIPHERTEXT_3).

```
message_3 =
(
  h'c4',
  h'de4a833d48b66474142cc9bdce87d93af835579c2dbf1b9e2fb4dc66600dbac6bb3cc0
  5c290ef35d515b4d7d6483f5096143b55644cfafd1ffaa7f2ba3863657831dd2e5bd0404
  3860140dc8'
)
```

Which encodes to the following byte string:

message_3 (CBOR Sequence) (80 bytes)
```
41 c4 58 4c de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87 d9 3a f8 35 57 9c 2d bf 1b
 9e 2f b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3 5d 51 5b 4d 7d 64 83 f5 09 61 4
3 b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57 83 1d d2 e5 bd 04 04 38 60 14 0d c8
```

C.1.5.5. OSCORE Security Context Derivation

From the previous message exchange, the Common Security Context for OSCORE [RFC8613] can be derived, as specified in Section 3.3.1.

First af all, TH_4 is computed: TH_4 = H(TH_3, CIPHERTEXT_3), where the input to the hash function is the CBOR Sequence of TH_3 and CIPHERTEXT_3

(TH_3, CIPHERTEXT_3)
(CBOR Sequence) (112 bytes)
```
58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a
79 07 f3 e7 85 43 67 fc 22 58 4c de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87
d9 3a f8 35 57 9c 2d bf 1b 9e 2f b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3
5d 51 5b 4d 7d 64 83 f5 09 61 43 b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57
83 1d d2 e5 bd 04 04 38 60 14 0d c8
```

And from there, compute the transcript hash TH_4 = SHA-256(TH_3, CIPHERTEXT_3)

TH_4 value (32 bytes)
```
51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a b4 d3 8c 34 81 96 09 ee 0d
5c 9d a6 e9 80 7f e5
```

When encoded as a CBOR bstr, that gives:

```
TH_4 (CBOR-encoded) (34 bytes)
58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a b4 d3 8c 34 81 96 09
ee 0d 5c 9d a6 e9 80 7f e5
```

To derive the Master Secret and Master Salt the same HKDF-Expand (PRK, info, L) is used, with different info and L.

For Master Secret:

L for Master Secret = 16

```
Info for Master Secret =
[
  "OSCORE Master Secret",
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'51ed3932bcbae8901c1d4deb94bd673ab4d38c34819609ee0d5c9da6e9
  807fe5' ]
]
```

When encoded as a CBOR bstr, that gives:

```
info (OSCORE Master Secret) (CBOR-encoded) (68 bytes)
84 74 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 65 63 72 65 74 83 f6 f6
f6 83 f6 f6 f6 83 18 80 40 58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd
67 3a b4 d3 8c 34 81 96 09 ee 0d 5c 9d a6 e9 80 7f e5
```

Finally, the Master Secret value computed is:

```
OSCORE Master Secret (16 bytes)
09 02 9d b0 0c 3e 01 27 42 c3 a8 69 04 07 4c 0e
```

For Master Salt:

L for Master Secret = 8

```
Info for Master Salt =
[
  "OSCORE Master Salt",
  [ null, null, null ],
  [ null, null, null ],
  [ 64, h'', h'51ed3932bcbae8901c1d4deb94bd673ab4d38c34819609ee0d5c9da6e98
  07fe5' ]
]
```

When encoded as a CBOR bstr, that gives:

info (OSCORE Master Salt) (CBOR-encoded) (66 bytes)
84 72 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 61 6c 74 83 f6 f6 f6 83
f6 f6 f6 83 18 40 40 58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a
b4 d3 8c 34 81 96 09 ee 0d 5c 9d a6 e9 80 7f e5

Finally, the Master Secret value computed is:

OSCORE Master Salt (8 bytes)
81 02 97 22 a2 30 4a 06

The Client's Sender ID takes the value of C_V:

Client's OSCORE Sender ID (1 bytes)
c4

The Server's Sender ID takes the value of C_U:

Server's OSCORE Sender ID (1 bytes)
c3

The algorithms are those negociated in the cipher suite:

AEAD Algorithm
10

HMAC Algorithm
5

C.2. Test Vectors for EDHOC Authenticated with Symmetric Keys (PSK)

Symmetric EDHOC is used:

method (Symmetric Authentication)
1

CoAP is used as transport:

corr (Party U is CoAP client)
1

No unprotected opaque application data is sent in the message exchanges.

The pre-defined Cipher Suite 0 is in place both on Party U and Party V, see Section 3.1.

C.2.1. Input for Party U

The following are the parameters that are set in Party U before the first message exchange.

Party U's ephemeral private key (32 bytes)
f4 0c ea f8 6e 57 76 92 33 32 b8 d8 fd 3b ef 84 9c ad b1 9c 69 96 bc 27 2a f1 f6 48 d9 56 6a 4c

Party U's ephemeral public key (value of X_U) (32 bytes)
ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f 58 88 97 cb 57 49 61 cf a9 80 6f

Connection identifier chosen by U (value of C_U) (1 bytes)
c1

Pre-shared Key (PSK) (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de

kid value to identify PSK (1 bytes)
a1

So ID_PSK is defined as the following:

```
ID_PSK =
{
  4:  h'a1'
}
```

This test vector uses COSE_Key objects to store the pre-shared key.

Note that since the map for ID_PSK contains a single 'kid' parameter, ID_PSK is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 5.1):

ID_PSK (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a1

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a1

C.2.2. Input for Party V

The following are the parameters that are set in Party U before the first message exchange.

Party V's ephemeral private key (32 bytes)
d9 81 80 87 de 72 44 ab c1 b5 fc f2 8e 55 e4 2c 7f f9 c6 78 c0 60 51 81 f3
7a c5 d7 41 4a 7b 95

Party V's ephemeral public key (value of X_V) (32 bytes)
fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94
6f 6b 09 a9 cb dc 06

Connection identifier chosen by V (value of C_V) (1 bytes)
c2

Pre-shared Key (PSK) (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de

kid value to identify PSK (1 bytes)
a1

So ID_PSK is defined as the following:

```
ID_PSK =
{
  4:  h'a1'
}
```

This test vector uses COSE_Key objects to store the pre-shared key.

Note that since the map for ID_PSK contains a single 'kid' parameter, ID_PSK is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 5.1):

ID_PSK (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a1

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a1

C.2.3. Message 1

From the input parameters (in Appendix C.2.1):

TYPE (4 * method + corr)
5

suite
0

SUITES_U : suite
0

G_X (X-coordinate of the ephemeral public key of Party U) (32 bytes)
ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f 58 88 97 cb
57 49 61 cf a9 80 6f

C_U (Connection identifier chosen by U) (CBOR encoded) (2 bytes)
41 c1

kid_value of ID_PSK (CBOR encoded) (2 bytes)
41 a1

No UAD_1 is provided, so UAD_1 is absent from message_1.

Message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  5,
  0,
  h'ab2fca32898322c208fb2dab5048bd43c355c6430f588897cb574961cfa9806f',
  h'c1',
  h'a1'
)
```

message_1 (CBOR Sequence) (40 bytes)
05 00 58 20 ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f
58 88 97 cb 57 49 61 cf a9 80 6f 41 c1 41 a1

C.2.4. Message 2

Since TYPE mod 4 equals 1, C_U is omitted from data_2.

G_Y (X-coordinate of the ephemeral public key of Party V) (32 bytes)
fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94
6f 6b 09 a9 cb dc 06

C_V (Connection identifier chosen by V) (1 bytes)
c2

Data_2 is constructed, as the CBOR Sequence of the CBOR data items above.

```
  data_2 =
  (
    h'fc3b339367a5225d53a92d380323afd035d7817b6d1be47d946f6b09a9cbdc06',
    h'c2'
  )

data_2 (CBOR Sequence) (36 bytes)
58 20 fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4
7d 94 6f 6b 09 a9 cb dc 06 41 c2
```

From data_2 and message_1 (from Appendix C.2.3), compute the input to the transcript hash TH_2 = H(message_1, data_2), as a CBOR Sequence of these 2 data items.

```
( message_1, data_2 ) (CBOR Sequence)
(76 bytes)
05 00 58 20 ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f
58 88 97 cb 57 49 61 cf a9 80 6f 41 c1 41 a1 58 20 fc 3b 33 93 67 a5 22 5d
53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94 6f 6b 09 a9 cb dc 06 41
c2
```

And from there, compute the transcript hash TH_2 = SHA-256(message_1, data_2)

```
TH_2 value (32 bytes)
16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c
db 7b 07 de e1 70 ca
```

When encoded as a CBOR bstr, that gives:

```
TH_2 (CBOR-encoded) (34 bytes)
58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d
34 1c db 7b 07 de e1 70 ca
```

C.2.4.1. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the symmetric case, salt is the PSK:

```
salt (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de
```

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
d5 75 05 50 6d 8f 30 a8 60 a0 63 d0 1b 5b 7a d7 6a 09 4f 70 61 3b 4a e6 6c
5a 90 e5 c2 1f 23 11
```

From there, PRK is computed:

```
PRK (32 bytes)
aa b2 f1 3c cb 1a 4f f7 96 a9 7a 32 a4 d2 fb 62 47 ef 0b 6b 06 da 04 d3 d1
06 39 4b 28 76 e2 8c
```

Key K_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_2
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07de
  e170ca' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_2) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 16 4f 44 d8 56 dd 15 22 2f
a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c db 7b 07 de e1 70 ca
```

L is the length of K_2, so 16 bytes.

From these parameters, K_2 is computed:

```
K_2 (16 bytes)
ac 42 6e 5e 7d 7a d6 ae 3b 19 aa bd e0 f6 25 57
```

Nonce IV_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_2
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07de
  e170ca' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_2) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7
35 8d 34 1c db 7b 07 de e1 70 ca
```

L is the length of IV_2, so 13 bytes.

From these parameters, IV_2 is computed:

```
IV_2 (13 bytes)
ff 11 2e 1c 26 8a a2 a7 7c c3 ee 6c 4d
```

C.2.4.2. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that UAD_2 is omitted.

- empty protected header
- external_aad = TH_2
- empty plaintext, since UAD_2 is omitted

From the parameters above, the Enc_structure A_2 is computed.

```
A_2 =
[
  "Encrypt0",
  h'',
  h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07dee170ca'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2
02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c db 7b 07 de e1 70 ca

The key and nonce used are defined in Appendix C.2.4.1:

- key = K_2

- nonce = IV_2

Using the parameters above, the ciphertext CIPHERTEXT_2 can be computed:

CIPHERTEXT_2 (8 bytes)
ba 38 b9 a3 fc 1a 58 e9

C.2.4.3. message_2

From the parameter computed in Appendix C.2.4 and Appendix C.2.4.2, message_2 is computed, as the CBOR Sequence of the following items: (G_Y, C_V, CIPHERTEXT_2).

```
message_2 =
(
  h'fc3b339367a5225d53a92d380323afd035d7817b6d1be47d946f6b09a9cbdc06',
  h'c2',
  h'ba38b9a3fc1a58e9'
)
```

Which encodes to the following byte string:

message_2 (CBOR Sequence) (45 bytes)
58 20 fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4
7d 94 6f 6b 09 a9 cb dc 06 41 c2 48 ba 38 b9 a3 fc 1a 58 e9

C.2.5. Message 3

Since TYPE mod 4 equals 1, C_V is not omitted from data_3.

C_V (1 bytes)
c2

Data_3 is constructed, as the CBOR Sequence of the CBOR data item above.

```
data_3 =
(
  h'c2'
)

data_3 (CBOR Sequence) (2 bytes)
41 c2
```

From data_3, CIPHERTEXT_2 (Appendix C.2.4.2), and TH_2 (Appendix C.2.4), compute the input to the transcript hash TH_2 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

```
( TH_2, CIPHERTEXT_2, data_3 ) (CBOR Sequence) (45 bytes)
58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d
34 1c db 7b 07 de e1 70 ca 48 ba 38 b9 a3 fc 1a 58 e9 41 c2
```

And from there, compute the transcript hash TH_3 = SHA-256(TH_2 , CIPHERTEXT_2, data_3)

```
TH_3 value (32 bytes)
11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81
b5 2b 8a f5 66 d7 fe
```

When encoded as a CBOR bstr, that gives:

```
TH_3 (CBOR-encoded) (34 bytes)
58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89
54 81 b5 2b 8a f5 66 d7 fe
```

C.2.5.1. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the symmetric case, salt is the PSK:

```
salt (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de
```

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
d5 75 05 50 6d 8f 30 a8 60 a0 63 d0 1b 5b 7a d7 6a 09 4f 70 61 3b 4a e6 6c
5a 90 e5 c2 1f 23 11
```

From there, PRK is computed:

```
PRK (32 bytes)
aa b2 f1 3c cb 1a 4f f7 96 a9 7a 32 a4 d2 fb 62 47 ef 0b 6b 06 da 04 d3 d1
06 39 4b 28 76 e2 8c
```

Key K_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_3
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af5
  66d7fe' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_3) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 11 98 aa b3 ed db 61 b8 a1
b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81 b5 2b 8a f5 66 d7 fe
```

L is the length of K_3, so 16 bytes.

From these parameters, K_3 is computed:

```
K_3 (16 bytes)
fe 75 e3 44 27 f8 3a ad 84 16 83 c6 6f a3 8a 62
```

Nonce IV_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_3
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af5
  66d7fe' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_3) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28
52 89 54 81 b5 2b 8a f5 66 d7 fe
```

L is the length of IV_3, so 13 bytes.

From these parameters, IV_3 is computed:

```
IV_3 (13 bytes)
60 0a 33 b4 16 de 08 23 52 67 71 ec 8a
```

C.2.5.2. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that PAD_2 is omitted.

- empty protected header
- external_aad = TH_3
- empty plaintext, since PAD_2 is omitted

From the parameters above, the Enc_structure A_3 is computed.

```
A_3 =
[
  "Encrypt0",
  h'',
  h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af566d7fe'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_3 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9
e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81 b5 2b 8a f5 66 d7 fe
```

The key and nonce used are defined in Appendix C.2.5.1:

- key = K_3
- nonce = IV_3

Using the parameters above, the ciphertext CIPHERTEXT_3 can be computed:

CIPHERTEXT_3 (8 bytes)
51 29 07 92 61 45 40 04

C.2.5.3. message_3

From the parameter computed in Appendix C.2.5 and Appendix C.2.5.2, message_3 is computed, as the CBOR Sequence of the following items: (C_V, CIPHERTEXT_3).

```
message_3 =
(
  h'c2',
  h'5129079261454004'
)
```

Which encodes to the following byte string:

message_3 (CBOR Sequence) (11 bytes)
41 c2 48 51 29 07 92 61 45 40 04

C.2.5.4. OSCORE Security Context Derivation

From the previous message exchange, the Common Security Context for OSCORE [RFC8613] can be derived, as specified in Section 3.3.1.

First af all, TH_4 is computed: TH_4 = H(TH_3, CIPHERTEXT_3), where the input to the hash function is the CBOR Sequence of TH_3 and CIPHERTEXT_3

(TH_3, CIPHERTEXT_3)
(CBOR Sequence) (43 bytes)
58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89
54 81 b5 2b 8a f5 66 d7 fe 48 51 29 07 92 61 45 40 04

And from there, compute the transcript hash TH_4 = SHA-256(TH_3, CIPHERTEXT_3)

TH_4 value (32 bytes)
df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7 57 41 3f a7 b6 a9 cf 28 3d
db 4c d4 c1 fd e4 3c

When encoded as a CBOR bstr, that gives:

```
TH_4 (CBOR-encoded) (34 bytes)
58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7 57 41 3f a7 b6 a9 cf
28 3d db 4c d4 c1 fd e4 3c
```

To derive the Master Secret and Master Salt the same HKDF-Expand (PRK, info, L) is used, with different info and L.

For Master Secret:

L for Master Secret = 16

```
Info for Master Secret =
[
  "OSCORE Master Secret",
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'df7c9b06f5dc0ee8860b396c78c5beb757413fa7b6a9cf283ddb4cd4c1
  fde43c' ]
]
```

When encoded as a CBOR bstr, that gives:

```
info (OSCORE Master Secret) (CBOR-encoded) (68 bytes)
84 74 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 65 63 72 65 74 83 f6 f6
f6 83 f6 f6 f6 83 18 80 40 58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5
be b7 57 41 3f a7 b6 a9 cf 28 3d db 4c d4 c1 fd e4 3c
```

Finally, the Master Secret value computed is:

```
OSCORE Master Secret (16 bytes)
8d 36 8f 09 26 2d c5 52 7f e7 19 e6 6c 91 63 75
```

For Master Salt:

L for Master Secret = 8

```
Info for Master Salt =
[
  "OSCORE Master Salt",
  [ null, null, null ],
  [ null, null, null ],
  [ 64, h'', h'df7c9b06f5dc0ee8860b396c78c5beb757413fa7b6a9cf283ddb4cd4c1f
  de43c' ]
]
```

When encoded as a CBOR bstr, that gives:

info (OSCORE Master Salt) (CBOR-encoded) (66 bytes)
84 72 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 61 6c 74 83 f6 f6 f6 83
f6 f6 f6 83 18 40 40 58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7
57 41 3f a7 b6 a9 cf 28 3d db 4c d4 c1 fd e4 3c

Finally, the Master Secret value computed is:

OSCORE Master Salt (8 bytes)
4d b7 06 58 c5 e9 9f b6

The Client's Sender ID takes the value of C_V:

Client's OSCORE Sender ID (1 bytes)
c2

The Server's Sender ID takes the value of C_U:

Server's OSCORE Sender ID (1 bytes)
c1

The algorithms are those negociated in the cipher suite:

AEAD Algorithm
10

HMAC Algorithm
5

Acknowledgments

The authors want to thank Alessandro Bruni, Martin Disch, Theis Groenbech Petersen, Dan Harkins, Klaus Hartke, Russ Housley, Alexandros Krontiris, Ilari Liusvaara, Karl Norrman, Salvador Perez, Eric Rescorla, Michael Richardson, Thorvald Sahl Joergensen, Jim Schaad, Carsten Schuermann, Ludwig Seitz, Stanislav Smyshlyaev, Valery Smyslov, Rene Struik, and Erik Thormarker for reviewing and commenting on intermediate versions of the draft. We are especially indebted to Jim Schaad for his continuous reviewing and implementation of different versions of the draft.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

ace
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

A. Somaraju
Tridonic GmbH & Co KG
S. Kumar
Philips Research
H. Tschofenig
ARM Ltd.
W. Werner
Werner Management Services e.U.

October 31, 2016

Security for Low-Latency Group Communication
draft-somaraju-ace-multicast-02.txt

Abstract

Some Internet of Things application domains require secure group communication. This draft describes procedures for authorization, key management, and securing group messages. We specify the usage of object security at the application layer for group communication and assume that CoAP is used as the application layer protocol. The architecture allows the usage of symmetric and asymmetric keys to secure the group messages. The asymmetric key solution provides the ability to uniquely authenticate the source of all group messages and this is the recommended architecture for most applications. However, some applications have strict requirements on latency for group communication (e.g. in non-emergency lighting applications) and it may not always be feasible to use the secure source authenticated architecture. In such applications we recommend the use of dynamically generated symmetric group keys to secure group communications.

Table of Contents

1. Introduction

There are low latency group communication use cases that require securing communication between a sender, or a group of senders, and a group of receivers. In the lighting use case, a set of lighting nodes (e.g., luminaires, wall-switches, sensors) are grouped together into a single "Application Group" and the following three requirements need to be addressed:

1. Only authorized members of the application group must be able to read and process messages.

2. Receivers of group messages must be able to verify the integrity of received messages as being generated within the group.

3. Message communication and processing must happen with a low latency and in synchronous manner.

This document discusses a group communication security solution that satisfies these three requirements. As discussed in Section 4, we recommend the usage of an asymmetric key solution that allows unique source authentication of all group messages. However, in situations where the low latency requirements can not be met (e.g. in non-emergency lighting applications), the alternative architecture discussed in Section 3 based on symmetric keys is recommended.

2. Terminology

This document uses the following terms from [I-D.ietf-ace-actors]: Authorization Server, Resource Owner, Client, Resource Server. The terms 'sender' and 'receiver' refer to the application layer messaging used for lighting control; other communication interactions with the supporting infrastructure uses unicast messaging.

When nodes are combined into groups there are different layers of those groups with unique characteristics. For clarity we introduce terminology for three different groups:

Application Group:

An application group consists of the set of all nodes that have been configured to respond to a single application layer request. For example, a wall mounted switch and a set of luminaires in a single room might belong to a single group and the switch may be used to turn on/off all the luminaires in the group simultaneously

with a single button press. In the remainder of this document we will use GId to identify an application group.

Multicast Group:

A multicast group consists of the set of all nodes that subscribe to the same multicast IP address.

Security Group:

A security group consists of the set of all nodes that have been provisioned with the same keying material. All the nodes within a security group share a security association or a sequence of security associations wherein a single association specifies the keying material, algorithm-specific information, lifetime and a key ID.

Source-authenticated Security Group:

A source-authenticated security group consists of the set of receiver nodes that have been provisioned with the public verification keying material of all the sender nodes and the set of sender nodes that are provisioned with their unique private signing keying material. All the nodes within a source-authenticated security group share a security association or a sequence of security associations wherein a single association specifies the the public or private keying material, algorithm-specific information, lifetime and a key ID.

Typically, the four groups might not coincide due to the memory constraints on the devices and also security considerations. For instance, in a small room with windows, we may have three application groups: "room group", "luminaires close to the window group" and "luminaires far from the window group". However, we may choose to use only one multicast group for all devices in the room and one security group for all the devices in the room. Note that every application group belongs to a unique security group. However, the converse is not always true. This implies that the application group ID maybe used to determine the associated security group but not vice versa.

The fact that security groups may not coincide with application groups implies that

(1) an application must be able to specify which resources on a resource server are accessible by a client that has access to the group key, and

(2) a method is required to associate the group key to the application group(s) for which the group key may be used.

In this document we provide fields that may be used to specify the "scope of the key" and "application groups for which the key may be used". A commissioner has a lot of flexibility to assign nodes to multicast groups and to security groups while the application groups will be determined by the semantics of the application itself. The exact partitioning of the nodes into security and multicast groups is therefore deployment specific.

3. Architecture - Group Authentication

Each node in a lighting application group might be a sender, a receiver or both sender and receiver (even though in Figure 1, we show nodes that are only senders or only receivers for clarity). The low latency requirement implies that most of the communication between senders and receivers of application layer messages is done using multicast IP. On some occasions, a sender in a group will be required to send unicast messages to unique receivers within the same group and these unicast messages also need communication security.

Two logical entities are introduced and they have the following function:

Key Distribution Center (KDC): This logical entity is responsible for generating symmetric keys and distributing them to the nodes authorized to receive them. The KDC ensures that nodes belonging to the same security group receive the same key and that the keys are renewed based on certain events, such as key expiry or change in group membership.

Authorization Server (AS): This logical entity stores authorization information about devices, meta-data about them, and their roles in the network. For example, a luminaire is associated with different groups, and may have meta-data about its location in a building.

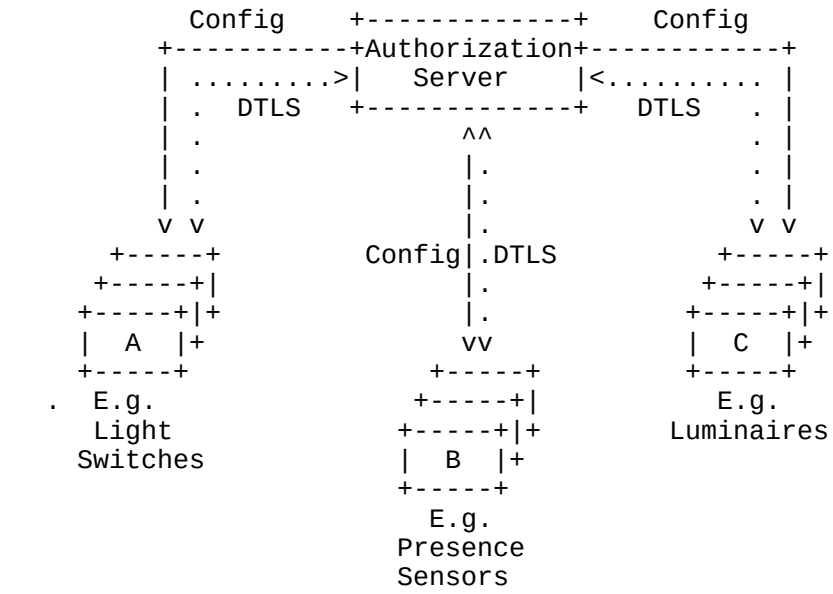
Note that we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 1 and Figure 2 provide an architectural overview. The dotted lines illustrate the use of unicast DTLS messages for securing the message exchange between all involved parties. The secured group messages between senders and receivers are indicated using lines with

star/asterisk characters. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) which are to be defined.

Figure 1 illustrates the information flow between an authorization server and the nodes participating in the lighting network, which includes all nodes that exchange lighting application messages. This step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control lights, may require an authorization server that dynamically executes access control decisions.

Figure 1 shows the commissioning phase where the nodes obtain configuration information, which includes the AT-KDC. The AT-KDC is an access token and includes authorization claims for consumption by the key distribution center. We use the access token terminology from [RFC6749]. The AT-KDC in this architecture may be a bearer token or a proof-of-possession (PoP) token. The bearer token concept is described in [RFC6750] and the PoP token concept is explained in [I-D.ietf-oauth-pop-architecture]. The AT-KDC is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-KDC is protected against modifications using a digital signature or a message authentication code. It is verified in Figure 2 by the KDC.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-KDC) and other meta-data.

Figure 1: Architecture: Commissioning Phase.

In the simplified message exchange shown in Figure 2 a sender requests a security group key and the access token for use with the receivers (called AT-R). The request contains information about the resource it wants to access, such as the application group and other resource-specific information, if applicable, and the previously obtained AT-KDC access token. Once the sender has successfully obtained the requested information it starts communicating with receivers in that group using group messages. The symmetric key obtained from the KDC is used to secure the groups messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary group key to verify the incoming messages and to determine what resource the requestor is authorized to access. Both pieces of information can be found in the AT-R access token.

Group messages need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using

a keyed message digest in combination with the group key. The use of symmetric keys is envisioned in this specification due to latency requirements. For unicast messaging between the group members and the AS or KDC, we assume the use of DTLS for transport security. However, the use of TLS, and application layer security is possible but is outside the scope of this document.

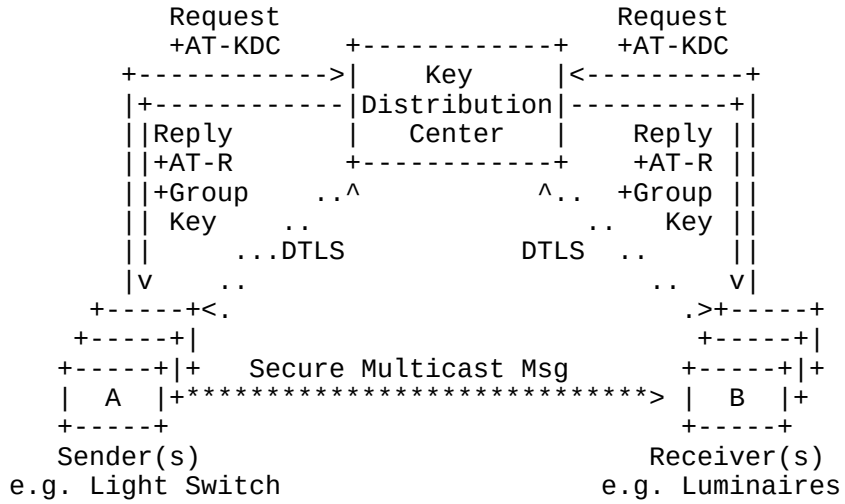


Figure 2: Architecture: Group Key Distribution Phase.

3.1. Assumptions

1. The AT-KDC is a manifestation of the authorization granted to a specific client (or user running a client). The AT-KDC is longer-lived and can be used to request multiple AT-Rs.

2. Each AT-R is valid for use with one or multiple application groups.

3. The AS and the KDC logical roles may reside in different physical entities.

4. The AT-KDC as well as the AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.

5. The AT-KDC token is opaque to the client. Data that is meant for processing by the client has to be conveyed to the client

separately. The AT-R token on the other hand is meant for consumption by the client.

6. The client requests AT-Rs for different application groups by including additional information in the request to the KDC for what application groups the AT-R(s) have to be requested. The KDC may return multiple AT-Rs in a single response (for performance reasons).

7. The AT-KDC and the AT-R are encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

3.2. AT-KDC Access Tokens

The AT-KDC contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Recipient/Audience: Indication to whom the AT-KDC was issued to. In this case, it is the KDC.

5. Client ID: Information about the client that was authenticated by the authorization server.

6. Issued at: Indicates date and time when the AT-KDC was created by the authorization server.

3.3. AT-R Access Tokens

Clients send the AT-KDC to the KDC in order to receive an AT-R.

The KDC MUST maintain a table consisting of scope values, which includes the application group id. These entries point to a sequence of security associations. A security association specifies the key material, algorithm-specific information, lifetime and a key ID and the key ID may be used to identify this security association.

The AS/KDC must guarantee the uniqueness of the client ids for its nodes. This may be accomplished by the AS/KDC assigning values to the nodes or by using information that is already unique per device (such as an EUI-64).

The KDC furthermore needs to be configured with information about the authorization servers it trusts. This may include a provisioned trust anchor store, or shared credentials (similar to a white list).

The KDC MUST generate new group keys after the validity period of the current group key expires.

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Security Group Key: Key to use for the group communication.

5. Algorithm: Used for secure group communication.

6. KID: Sequentially increasing ID of the key for the security group (the devices may store an older key to help with key rolling.)

7. Issued at: Indicates date and time when the AT-R was created by the KDC.

3.4. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

    * Client ID (unencrypted, integrity protected): Every sender managed by a key distribution center MUST have a unique client ID.

   * Sequence Number (unencrypted, integrity protected): Used for replay protection.

   * An implicit IV that is either derived from the keys at the end-points or fixed to a certain value by standard (not sent in the message)

2. MAC (not integrity protected): For integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.

2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.

3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.

4. Key ID (unencrypted, integrity protected): Is used to select the current security association from the sequence of security associations identified by the application group id.

5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)

6. CoAP Payload (encrypted, integrity protected).

3.5. Receiver Algorithm

All receiving devices MUST maintain a table consisting of mappings of application group id, to a sequence of security associations.

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association.

The key id is used for situations where the group key is updated by the KDC (for example in situations where a device in a group is lost or stolen).

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- Application Group ID lookup does not return any security association.
- Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- Integrity check fails.
- Decryption fails.
- Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

3.6. Sender Algorithm

Figure 3 describes the algorithm for obtaining the necessary credentials to transmit a secure group message. When the sender wants to send a message to the application group, it checks if it has the respective group key. If no group key is available then it determines whether it has an access token for use with the KDC (i.e., AT-KDC). If no AT-KDC is found in the cache then it contacts the authorization server to obtain that AT-KDC. Note that this assumes

that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported. In the other case where the AT-KDC is already available the sender contacts the KDC to obtain a group key. If a group key is already available then the sender can transmit a secured message to the group immediately.

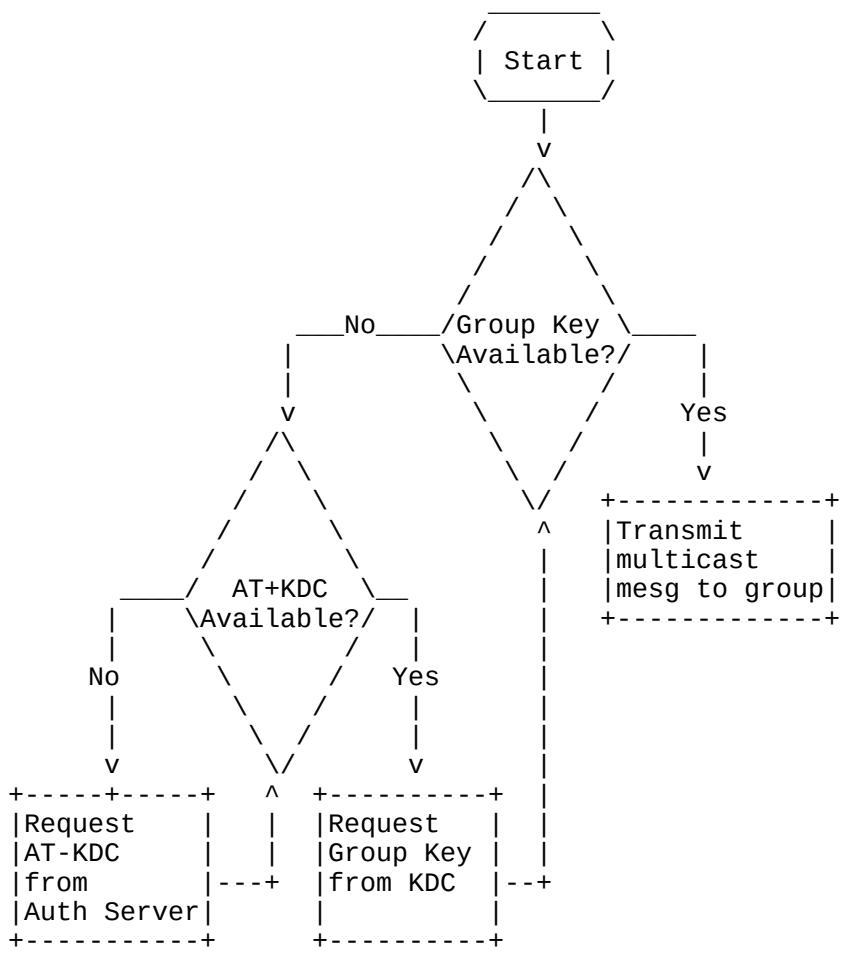


Figure 3: Steps to Transmit Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a group key; the sender is likely to be

pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

Group keys have a lifetime, which is configuration-dependent, but mechanisms need to be provided to update the group keys either via the sender asking for a group key renewal or via the KDC pushing new keys to senders and receivers. The lifetime can be based on time or on the number of transmitted messages.

4. Architecture - source authentication

This section discusses the usage of asymmetric keys to achieve source authentication of group messages and is the recommend architecture for securing group messages. However, this solution may not meet the low latency requirement without adequate hardware support but still most of the group communication between senders and receivers of application layer messages is done using multicast IP.

Unlike the previous architecture, the current architecture requires only the Authorization Server (AS) logical entity as defined in the previous section.

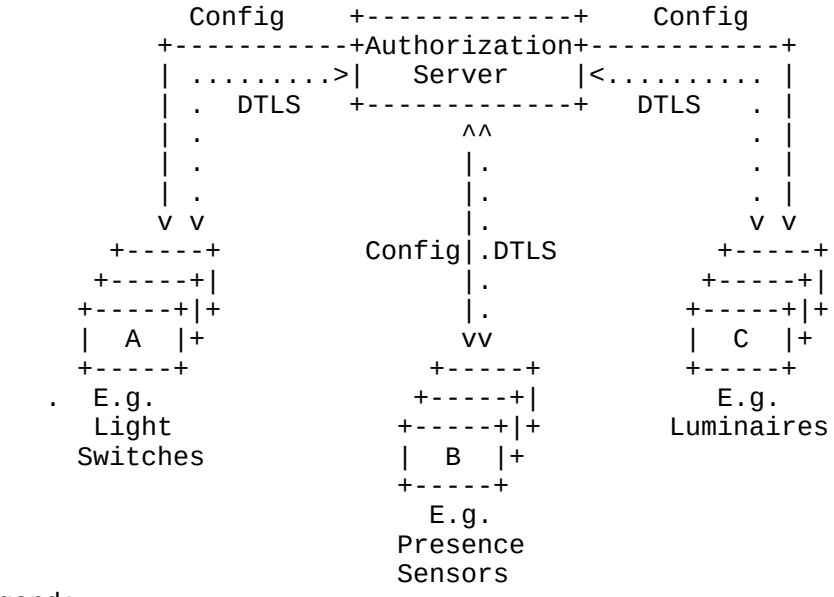
As in the previous case we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 4 and Figure 5 provide an architectural overview for the source authenticated case. The main differences from the previous case is that the AS provides directly the AT-R tokens. Further no KDC is required in this case since the senders and receivers can use their public-private key pair credentials to secure messages. The AS may provide authorization based on the pre-existing device credentials or issue new credentials to the devices. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) but based on public key signatures which are to be defined.

Figure 4 illustrates the information flow between an authorization server and the nodes participating in the source-authenticated group network. Like the previous case, this step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control

lights, may require an authorization server that dynamically executes access control decisions.

Figure 4 shows the commissioning phase where the nodes obtain configuration information, which includes directly the AT-R. The AT-R is an access token and includes authorization claims for consumption by the receivers. The AT-R may be a bearer token or a proof-of-possession (PoP) token. The AT-R is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-R is protected against modifications using a digital signature. It is verified in Figure 5 by the receivers.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-R) and other meta-data.

Figure 4: Architecture - Source-authenticated: Commissioning Phase.

In the simplified message exchange shown in Figure 5 a sender starts communicating with receivers in that source-authenticated group using public-key signed group messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary public verification key of the senders (or a root verification key if they are certified by the same authority) to verify the incoming messages and the public verification key of the AS to determine what resource the requestor is authorized to access. Both pieces of information can either be found in the AT-R access token or separately configured during the commissioning phase.

Source-authenticated Group messages also need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using a public-key signature. This may not achieve the latency requirements and used where source-authentication is more important. For unicast messaging between the group members and the AS , we assume the use of DTLS for transport security.

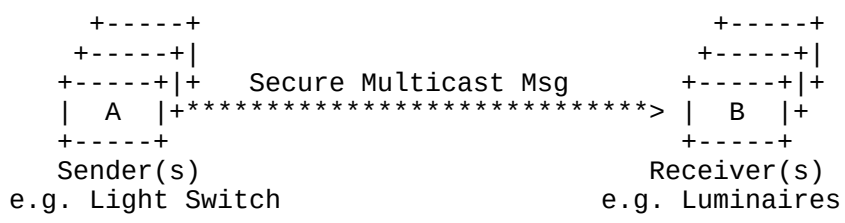


Figure 5: Architecture - Source-authenticated: Group communication.

4.1. Assumptions

1. The AT-R is a manifestation of the authorization granted to a specific client (or user running a client). The AT-R is longer-lived and can be used directly for source-authenticated group communication until it is revoked or expired.

2. Each AT-R is valid for use with one or multiple application groups.

3. The AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.

4. The AT-R token is not opaque to the client and is meant for consumption by the client.

5. The client requests AT-Rs for different application groups by including additional information in the request to the AS for what application groups the AT-R(s) have to be requested. The AS

may return multiple AT-Rs in a single response (for performance reasons).

6. The AT-R is encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

4.2. AT-R Access Tokens

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Recipient/Audience: Indication to whom the AT-R was issued to. In this case, it is the receivers.

5. Client ID: Information about the client that was authenticated by the authorization server.

6. Client public key: The public key to use for signing the source-authenticated group communication. These public key may be optionally certified using the AS key or a domain root key. This reduces the need for additional per-device public key storage on the receivers.

7. Algorithm: Used for source-authenticated secure group communication.

8. Issued at: Indicates date and time when the AT-R was created by the authorization server.

4.3. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

   * Client ID (unencrypted, integrity protected): Every sender managed by the AS MUST have a unique client ID.

   * Sequence Number (unencrypted, integrity protected): Used for replay protection.

2. Signature (not integrity protected): For source-authenticated integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.

2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.

3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.

4. Key ID (unencrypted, integrity protected): Is used to select the correct security association containing the verification key from the sequence of security associations identified by the application group id.

5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)

6. CoAP Payload (encrypted, integrity protected).

4.4. Receiver Algorithm

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association to use to verify the message.

The key id is used for situations where the client may have different keys for different applications.

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the

request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- Application Group ID lookup does not return any security association.
- Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- Integrity check fails.
- Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

4.5. Sender Algorithm

Figure 6 describes the algorithm for obtaining the necessary credentials to transmit a source-authenticated secure group message. When the sender wants to send a message to the application group, it checks if it has the respective signing key that matches the KID in the AT-R. If no signing key is available then it contacts the authorization server to obtain the AT-R and corresponding signing keys. Note that this assumes that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported.

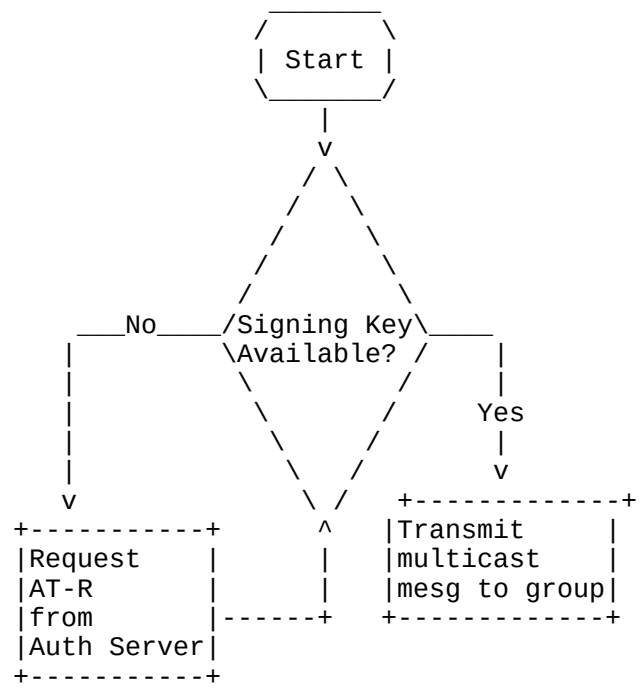


Figure 6: Steps to Transmit Source-authenticated Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a AT-R; the sender is likely to be pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

5. Security Considerations

5.1. Applicability statement

This document describes two architectures based on symmetric group keys in Section 3 and asymmetric keys in Section 4.

The symmetric key solution is based on a group key that is shared between all group members including senders and receivers. As all members of the group posses the same key, it is only possible to authenticate group membership for the source of a message. In particular, it is not possible to authenticate the unique source of a message and consequently it is not possible to authorize a single

node to control a group. Moreover, because the group key is shared across multiple nodes, it may be easier for an attacker to determine the group key by attacking any member of the group (note that this group key is dynamically generated and is usually stored in volatile memory which offers some addition protection). Subsequent to such an attack, it is also difficult to determine which of the group members was compromised and this makes it difficult to return the system to normal operation after an attack.

The asymmetric key solution distinguishes between a sender in the group and the receivers. In particular, the sender is in possession of a private key and the receivers are in possession of the corresponding public key. This allows the unique source of any group message to be authenticated. Moreover, an attacker cannot compromise the system by breaking into any of the receiving nodes. However, for constrained devices, the asymmetric key solution comes at a processing cost with cryptographic computations taking too long.

Therefore, it is recommended that whenever possible, the architecture with source authentication SHOULD be used to secure all multicast communication. However, in less sensitive applications (e.g. controlling luminaires in non-emergency applications), the architecture without source authentication MAY be used. When using the symmetric key solution two mitigating factors could improve system security. It is possible to achieve source authentication of messages at lower layers by requiring unique MAC layer keys for all devices within the network. The symmetric group keys are dynamically generated and therefore SHOULD be stored in volatile memory.

5.2. Token Verification

Due to the low latency requirements, token verification needs to be done locally and cannot be outsourced to other parties. For this reason a self-contained token must be used and the receivers are required to follow the steps outlined in Section 7.2 of RFC 7519 [RFC7519]. This includes the verification of the message authentication code protecting the contents of the token and the encryption envelope protecting the contained symmetric group key.

5.3. Token Revocation

Tokens have a specific lifetime. Setting the lifetime is a policy decision that involves making a trade-off decision. Allowing a longer lifetime increases the need to introduce a mechanism for token revocation (e.g., a real-time signal from the KDC/Authorization Server to the receivers to blacklist tokens) but lowers the communication overhead during normal operation since new tokens need to be obtained only from time to time. Real-time communication with

the receivers to revoke tokens may not be possible in all cases either, particularly when off-line operation is demanded or in small networks where the AS or even the KDC is only present during commissioning time.

We therefore recommend to issue short-lived tokens for dynamic scenarios like users accessing the lighting infrastructure of buildings using smartphones, tablets and alike to avoid potential security problems when tokens are leaked or where authorization rights are revoked. For senders that are statically mounted (like traditional light switches) we recommend a longer lifetime since re-configurations and token leakage is less likely to happen frequently.

To limit the authorization rights, tokens should contain an audience restriction, scoping their use to the intended receivers and to their access level.

5.4. Time

Senders and receivers are not assumed to be equipped with real-time clocks but these devices are still assumed to interact with a time server. The lack of accurate clocks is likely to lead to clock drifts and limited ability to check for replays. For those cases where no time server is available, such as in small network installations, token verification cannot check for expired tokens and hence it might be necessary to fall-back to tokens that do not expire.

6. Operational Considerations

6.1. Persistence of State Information

Devices in the lighting system can often be powered down intentionally or unintentionally. Therefore the devices may need to store the authorization tokens and cryptographic keys (along with replay context) in persistent storage like flash. This is especially required if the authorization server is no more online because it was removed after the commissioning phase. However the decision on the data to be persistently stored is a trade-off between how soon the devices can be back online to normal operational mode and the memory wear caused due to limited program-erase cycles of flash over the 15-20 years life-time of the device.

The different data that may need to be stored are access tokens AT-KDC, AT-R and last seen replay counter.

6.2. Provisioning in Small Networks

In small networks the authorization server and the KDC may be available only temporarily during the commissioning process and are not available afterwards.

6.3. Client IDs

A single device should not be managed by multiple KDCs. However, a group of devices in a domain (such as a lighting installation within an enterprise) should either be managed by a single KDC or, if there are multiple KDCs serving the devices in a given domain, these KDCs MUST exchange information so that the assigned client id and application group id values are unique within the devices in that domain. We assume that only devices within a given domain communicate with each other using group messages.

6.4. Application Groups vs. Security Groups

Multiple application groups may use the same key for performance reasons, reducing the number of keys needed to be stored - leading to less RAM needed by each node. This is only a reasonable option if the attack surface is not increased. For example, a room A is configured to use three application groups to address a subset of the device. In addition to configuring all nodes in room A with these three application groups the nodes are configured with a special group that allows them to access all devices in room A, referred as the all-nodes-in-room-A group. In this case, having the nodes to use the same key for the all-nodes-in-room group and the three groups does not increase the attack surface since any node can already use the all-nodes-in-room-A group to control other devices in that room. The three application groups in room A are a subset of the larger all-nodes-in-room-A group.

6.5. Lost/Stolen Device

The following procedure MUST be implemented if a device is stolen or keys are lost.

1. The AS tells the KDC to invalidate the AT-KDC.

2. The KDC no longer returns a new group key if the invalidated AT-KDC is presented to it.

3. The KDC generates new keys for all security groups to which the compromised device belongs.

The KDC SHOULD inform all devices in the security group to update their group key. This requires the KDC to maintain a list of all devices that belong to the security group and to be able to contact them reliably.

7. Acknowledgements

The author would like to thank Esko Dijk for his help with this document.

Parts of this document are a byproduct of the OpenAIS project, partially funded by the Horizon 2020 programme of the European Commission. It is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of fitness for a particular purpose. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the OpenAIS project or the European Commission.

8. IANA Considerations

This document defines one CoAP Header Option Application Group ID that MUST be allocated in the Registry "CoAP Option Numbers" of [RFC6749]. IANA is requested to allocation TBD option number to application group ID in this specification.

9. References

9.1. Normative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-04 (work in progress), September 2016.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-23 (work in progress), October 2016.

[I-D.wahlstroem-ace-cbor-web-token]
Wahlstroem, E., Jones, M., and H. Tschofenig, "CBOR Web Token (CWT)", draft-wahlstroem-ace-cbor-web-token-00 (work in progress), December 2015.

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

9.2. Informative References

[I-D.ietf-oauth-pop-architecture] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-06 (work in progress), October 2016.

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

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

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

Appendix A. Access Levels

A characteristic of the lighting domain is that access control decisions are also impacted by the type of operation being performed and those categories are listed below. The following access levels are pre-defined.

Level 0: Service detection only

This is a service that is used with broadcast service detection methods. No operational data is accessible at this level.

Level 1: Reporting only

This level allows access to sensor and other (relatively uncritical) operational data and the device error status. The operation of the system cannot be influenced using this level.

Level 2: Standard use

This level allows access to all operational features, including access to operational parameters. This is the highest level of access that can be obtained using (secure) multicast.

Level 3: Commissioning use / Parametrization Services

This level gives access to certain parameters that change the day-to-day operation of the system, but does not allow structural changes.

Level 4: Commissioning use / Localization and Addressing Services

(including Factory Reset) This level allows access to all services and parameters including structural settings.

Level 5: Software Update and related Services

This level allows the change and upgrade of the software of the devices.

Note: The use of group security is disallowed for level higher than Level 2 and unicast communication is used instead.

Authors' Addresses

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn 6850
Austria

Email: abhinav.somaraju@tridonic.com

Sandeep S. Kumar
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
Netherland

Email: ietf.author@sandeep-kumar.org

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Walter Werner
Werner Management Services e.U.
Josef-Anton-Herrburgerstr. 10
Dornbirn 6850
Austria

Email: werner@werner-ms.at