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.

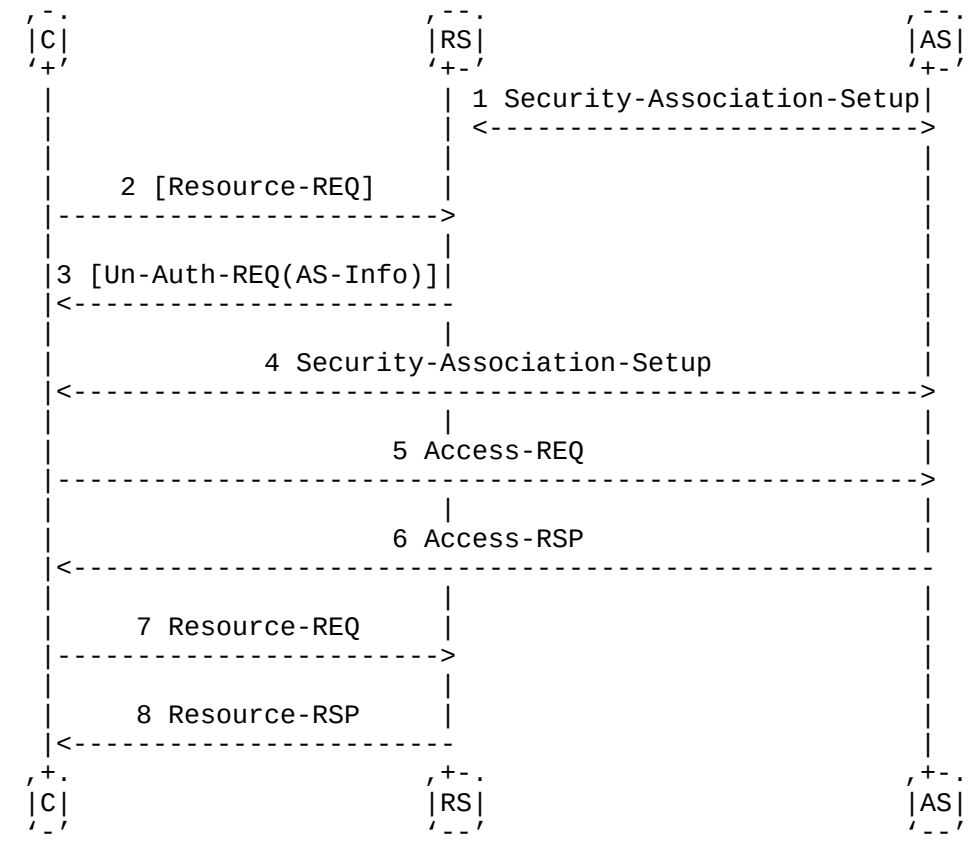

Figure 1: PAT protocol message flow

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:

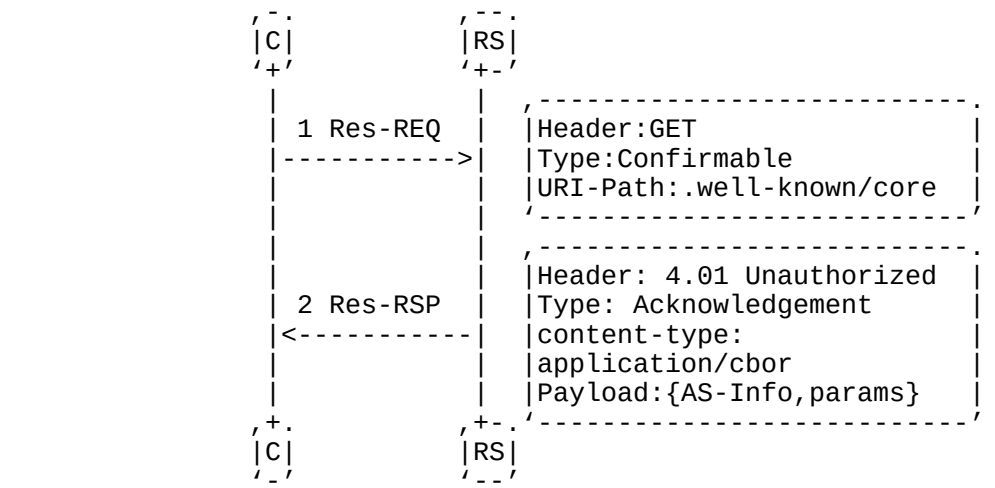

Figure 2: C<->RS Resource-Request and Unauthorized as response

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.

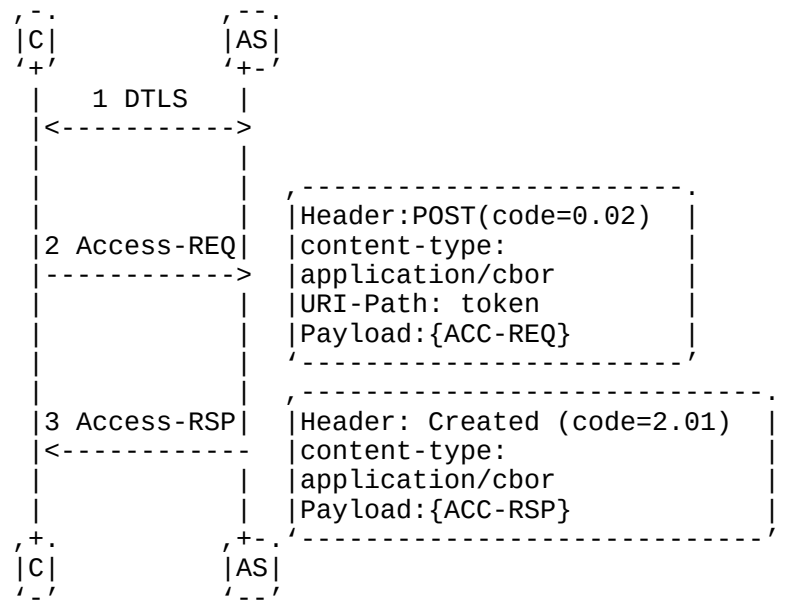

Figure 5: Access-Request and Access-Response

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